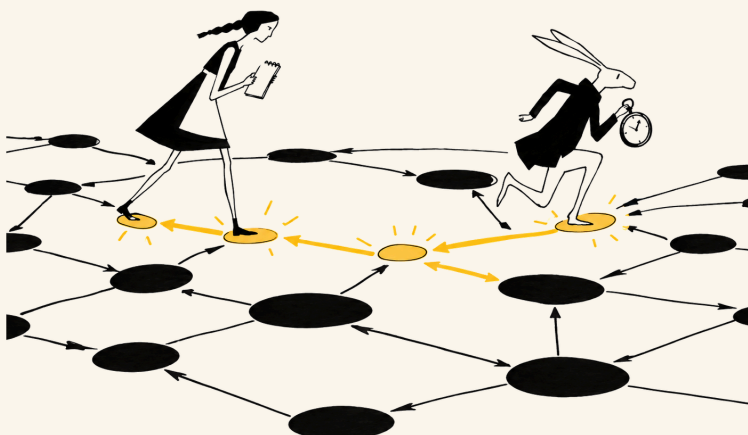


Приключения Алисы в Стране дифференцируемых чудес



Simone Scardapane · 2024

Иллюстрированные аудиолекции

Lecturized

Приключения Алисы в Стране дифференцируемых чудес

Иллюстрированные аудиолекции

Simone Scardapane · 2024

Пересказано в иллюстрированных лекциях Lecturized

lecturized.com

По оригиналу: Alice's Adventures in a Differentiable Wonderland, Volume I: A Tour of
the Land

Содержание

Несколько деталей, масштаб без предела

Старые идеи, новая мощь

Коробки с числами и их формы

Стрелки, которые смотрят в одну сторону

Трафареты, пакеты и машины из чисел

Сколько работы в одном умножении

Рецепт из букв и наклон кривой

Наклоны во всех направлениях

Спуск с горы в тумане

Где может закончиться прогулка

Шарик с инерцией

От учебного кода к настоящим данным

Обучение без меток и повторное использование выученного

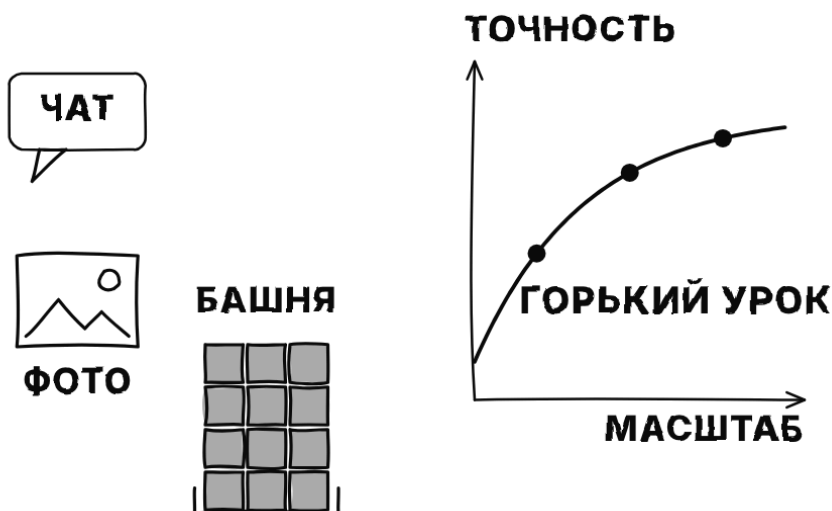
Оценка догадки

Запоминание — не обучение

Самая правдоподобная модель

Несколько деталей, масштаб без предела

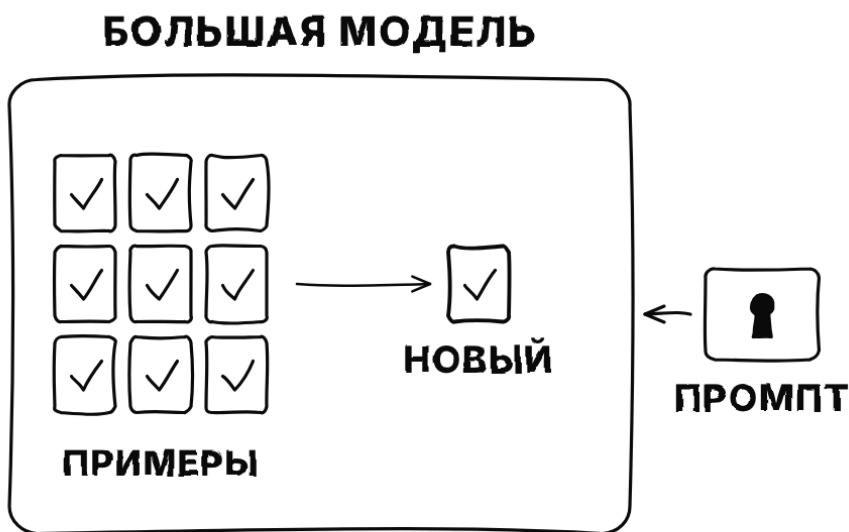
Откройте чат-ассистент на телефоне и попросите его написать стихотворение ко дню рождения вашей тёти, а потом откройте приложение с фотографиями и наберите в поиске «собака на пляже». Две совсем разные задачи, одна со словами, другая с картинками, и обе решены за секунду. Вы могли бы ожидать, что за ними стоят две совершенно разные машины, каждая со своим секретным рецептом. Неожиданность, с которой начинается книга, в обратном: обе они — нейронные сети, и обе собраны из очень небольшого набора одних и тех же простых деталей. За последние годы изменились не столько детали, сколько масштаб, в котором их складывают и обучают.



Те же детали, выше башни

Книга указывает на идею, которую называют законами масштабирования. Грубо говоря, почти для любой задачи, если дать модели больше данных, больше вычислительной мощности и больший размер, всё сразу, её точность растёт так, что это можно предсказать заранее. Если сказать наоборот: вычислительная мощность, нужная для заданной точности, со временем падает в постоянное число раз. Вот почему компании вкладывают в искусственный интеллект огромные суммы: кривая была достаточно надёжной, чтобы на неё ставить. Исследователь Ричард Саттон дал этому уроку имя, горький урок: простые, универсальные инструменты в сочетании со всё большей вычислительной мощностью обычно побеждают хитрые ручные приёмы.

Если принять это буквально, звучит почти как волшебство, и книга внимательна к тому, где волшебство может не сработать. Сеть обучают подражать скрытой закономерности за примерами, которые ей показывают, как фальсификатор учит подпись по множеству образцов. Современные сети так велики, что могли бы просто запомнить каждый пример, как студент, который заучивает лист с ответами наизусть и ничего не понимает. Однако на практике обученные модели хорошо справляются со многими задачами, которых в их обучении вообще не было. По мере роста обучающих коллекций граница между знакомыми и незнакомыми вопросами размывается, и большие модели, похоже, на удивление мало опираются на чистое запоминание.



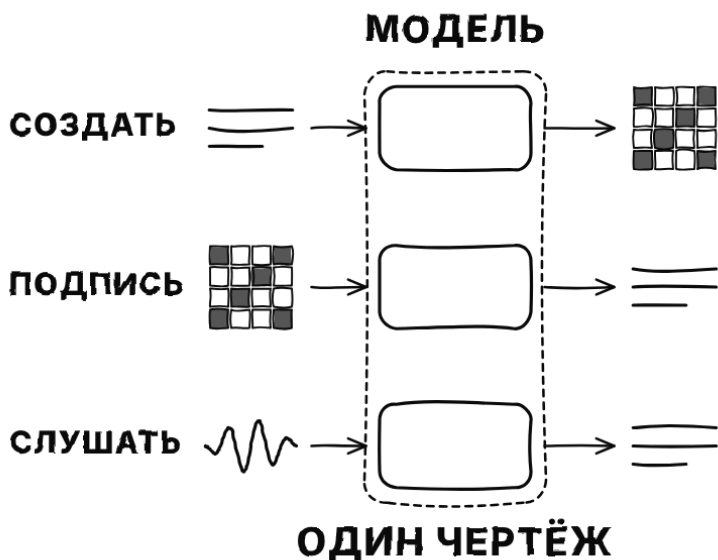
Запомнить или обобщить

Эти очень большие модели, которые можно приспособить ко многим последующим задачам, иногда называют фундаментальными моделями, и они изменили то, как люди пользуются сетями. Очень часто вы уже ничего не строите: вы набираете инструкцию, промпт, в модель, которую обучил кто-то другой, а её внутренности остаются закрытым ящиком. Книга сравнивает это с обычными программами. Вместо того чтобы писать каждую программу самому, вы устанавливаете готовую, чей исходный код никогда не видите; и как лишь немногие фирмы могут строить огромные деловые системы, так лишь немногие команды в мире имеют компьютеры и знания, чтобы обучить крупнейшие языковые модели, например семейство Llama.

Но у сравнения есть вторая половина, и в ней причина появления книги. Как открытый исходный код позволяет любому изменить или пересобрать программу, так обычное оборудование и немного изобретательности позволяют любому экспериментировать с этими моделями. Вы можете дообучить модель под свою задачу, объединить две модели, сжать одну так, чтобы она работала на маленьком устройстве, проверить, как легко она ломается, или придумать новую конструкцию. Для любого из этого нужно заглянуть под капот: как модель хранит и преобразует данные внутри, со всеми приёмами, которые добыты опытом и отладкой. Автор предлагает книгу как такую точку входа и надеется, что читатель, любопытный, как Алиса, получит удовольствие от путешествия.

Книга предполагает, что вы уже знаете основную идею обучения с учителем: собрать примеры нужного поведения, а потом обучать систему, пока она не начнёт им подражать. Эта простая

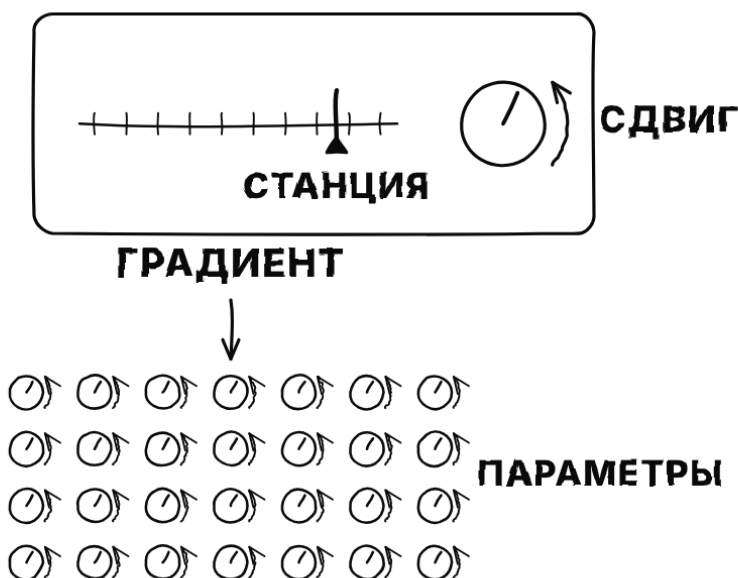
идея заводит на удивление далеко. Создание картинок по словам сводится к тому, чтобы собрать огромную кучу изображений с подписями; подражание английскому языку становится умением угадывать следующее предложение по предыдущим, так и обучают чат-ассистента; чтение рентгеновского снимка становится большой коллекцией снимков с заключениями врачей. Рисунок в книге сортирует задачи по тому, что входит и что выходит: генерация картинок превращает текст в сетку пикселей, подпись к картинке идёт в обратную сторону, а ответ на устный вопрос превращает звук в текст. Автора завораживает то, что модели для всех этих задач устроены очень похоже.



Что входит, что выходит

Опыт

Возьмите радио с одной ручкой настройки и незнакомую станцию, которую вы хотите найти. Чуть поверните ручку и послушайте: музыка стала чище или шумнее, чем секунду назад? Эта маленькая проверка подсказывает, в какую сторону крутить дальше, и после нескольких подвижек вы на станции, хотя никогда не знали, где она. Теперь представьте не одну ручку, а миллиарды, и все их подкручивают одновременно по одному правилу; это и есть обучение.



Подкрутите ручку, послушайте

Книга называет обучение поиском. Вы начинаете с программы, у которой огромное число подстраиваемых ручек, они называются параметрами, и подкручиваете их, пока программа не станет вести себя достаточно хорошо. При миллионах или даже триллионах ручек слепой перебор безнадежен, так что поиску нужен проводник. Дифференцируемые модели получают этого проводника потому, что собраны только из деталей, чей отклик на крошечный сдвиг можно вычислить точно. Знать для всех ручек сразу, что сделает крошечный поворот, — вот что делает автоматическую настройку возможной, и два инструмента за этим, автоматическое дифференцирование и градиентный спуск, занимают первую часть книги, со второй главы по шестую.

Если смотреть на сеть как на цепочку таких деталей, возникают два вопроса: какие данные могут входить и выходить и какие детали мы можем использовать? Приём с подкручиванием требует гладких величин, так что буквы или целые числа, которые скачут от одного значения к другому, напрямую использовать нельзя. Большие массивы чисел, которые книга называет тензорами, подходят идеально: фотография — ровно такой массив, и его можно перестраивать простыми сочетаниями прямолинейных и изгибающих шагов. Вторая часть, главы с седьмой по девятую, изучает одну знаменитую деталь, свёртку, которая работает со всем, что выложено по порядку: звук, картинки, текст и видео. По пути она показывает, как строить глубокие модели из многих шагов один за другим, и идеи нарезки текста на токены и порождения его кусок за куском, основу современных языковых моделей. Третья часть спускается глубже в кроличью нору: внимание и трансформеры для наборов элементов в главах десять и одиннадцать, графы в главе

двенадцать и рекуррентные слои для последовательностей во времени в главе тринадцать.

Запомним

Чат-ассистент и приложение с фотографиями на вашем телефоне — обе нейронные сети, собранные из одного и того же небольшого набора простых деталей. Законы масштабирования говорят, что больше данных, больше вычислений и большие модели предсказуемо дают лучшую точность, поэтому в это так много вкладывают, а Саттон назвал это горьким уроком. Огромные модели могли бы запомнить свои примеры, но они хорошо обобщают, и большинство людей теперь пользуется ими через промпты как закрытыми ящиками. Чтобы изменить или построить такие модели, нужно заглянуть внутрь, и взгляд книги таков: обучение — это поиск по бесчисленным ручкам, которым управляет отклик каждой детали на крошечный сдвиг. Затем книга переходит от этих инструментов к свёрткам для упорядоченных данных, к вниманию, графам и рекуррентным слоям.

Вопросы

1. Почему компании вкладывают такие огромные суммы в то, чтобы делать модели больше?

- a. Потому что большим моделям вообще не нужны обучающие данные
 - b. Потому что больше данных, вычислений и размера вместе предсказуемо давали лучшую точность
 - c. Потому что для каждой новой задачи нужен совершенно новый вид сети
 - d. Потому что только большие модели способны запоминать ответы
2. Огромная сеть могла бы просто запомнить каждый обучающий пример. Что, по словам книги, происходит на самом деле?
- a. Обученные модели хорошо справляются со многими задачами, которых в их обучении вообще не было
 - b. Большие модели лишь повторяют свои обучающие примеры слово в слово
 - c. Запоминание невозможно, потому что сети слишком малы, чтобы хранить примеры
 - d. Модели работают только на вопросах, идентичных их обучающим данным
3. Почему книга настаивает, что для дообучения, объединения или сжатия модели нужно заглянуть под капот?
- a. Потому что промпты для таких изменений запрещены
 - b. Потому что менять модели разрешено только крупнейшим компаниям
 - c. Потому что изменить модель значит понять, как она хранит и преобразует данные внутри

- d. Потому что у моделей с открытым кодом нет параметров для настройки
4. Почему буквы или целые числа нельзя подавать дифференцируемой модели напрямую?
- a. Потому что они занимают слишком много памяти
 - b. Потому что приём с подкручиванием требует гладких величин, а они скачут от одного значения к другому
 - c. Потому что тензоры могут хранить только фотографии
 - d. Потому что свёртки не принимают ничего, кроме звука

Ответы

1. **b)** Законы масштабирования говорят, что рост данных, вычислительной мощности и размера модели вместе предсказуемо повышает точность, так что ставка надёжна.
2. **a)** Хотя они достаточно велики, чтобы выучить лист с ответами наизусть, обученные модели хорошо обобщают и на удивление мало опираются на чистое запоминание.
3. **c)** Пользоваться моделью как закрытым ящиком достаточно для промптов, но чтобы приспособить или построить модель, нужно знать, как она обрабатывает данные внутри, со всеми выученными приёмами.
4. **b)** Обучение опирается на знание того, что делает крошечный сдвиг, а для этого нужны гладкие величины; буквы и целые числа дискретны, тогда как большие массивы чисел, например фотографии, подходят хорошо.

Задачи

1. Объясните своими словами, почему книга называет обучение поиском и почему этому поиску нужны дифференцируемые

детали, когда у модели миллиарды параметров.

Подсказка: Вспомните ручку радио: что говорит вам маленький поворот и что происходит, когда ручек миллиарды?

Ответ: Модель — это программа с огромным числом подстраиваемых ручек, а обучение означает подкручивать их, пока программа не станет вести себя хорошо. При миллиардах ручек случайный перебор никогда бы не закончился. Если каждая деталь откликается на крошечный сдвиг так, что это можно вычислить точно, мы знаем для всех ручек сразу, в какую сторону крутить, и автоматические инструменты, такие как градиентный спуск, могут выполнить настройку.

2. Книга говорит, что многие очень разные задачи становятся одним и тем же видом задачи при обучении с учителем.

Возьмите создание картинок по словам и объясните, что здесь «примеры» и чему учится модель.

Подсказка: Спросите себя, что входит, что должно выходить и какую коллекцию пар вы бы собрали.

Ответ: Вход — текст, а выход — изображение, так что примеры — это очень большая куча изображений с их подписями. Модель обучают до тех пор, пока по подписи она не начнёт выдавать подходящую картинку, подражая поведению, показанному в собранных парах.

3. Опираясь на сравнение книги с программами, объясните обе стороны: почему большинство людей теперь пользуется большими моделями как закрытыми ящиками и почему обычные люди всё равно могут с ними экспериментировать.

Подсказка: Одна сторона — о том, кто может построить крупнейшие модели; другая — об открытом исходном коде.

Ответ: Как и огромные деловые программы, крупнейшие модели могут построить лишь немногие команды с гигантскими компьютерами и знаниями, поэтому большинство людей просто пишет промпты модели, которую обучил кто-то другой. Но как открытый исходный код может изменить или пересобрать любой, так обычное оборудование и немного изобретательности позволяют людям дообучать, объединять, сжимать или проверять модели, если они понимают, что происходит внутри.

Старые идеи, новая мощь

Представьте газетную фотографию 1958 года: серьёзный мужчина в белой рубашке рядом со шкафом, полным проводов, под заголовком, который обещает машину, способную учиться, как мозг. Этот мужчина — американский психолог Фрэнк Розенблатт, а машина, перцептрон, была одной из первых работающих нейронных сетей. Книга показывает эту сцену нарочно, с подмигиванием: шумиха вокруг искусственного интеллекта, только на дворе 1958 год. Волнение, которое вы видите сегодня вокруг чат-ассистентов, не первая волна, а идеи под ней старше, чем думает большинство людей.

Само название «нейронная сеть» — остаток ранних попыток двадцатого века подражать нейронам живого мозга. Слова того времени остались с нами, например нейроны, слои и активации, и книга сохраняет их, чтобы совпадать с программами, которыми все пользуются. Интерес приходил волнами. Между 2012 и 2017 годами большие открытые соревнования, прежде всего конкурс ImageNet по распознаванию объектов на фотографиях, заставили сети быстро расти в размере и сложности. Вторая волна началась в 2017 году с трансформеров, которые за очень короткое время захватили область языка, как несколькими годами раньше одно семейство моделей захватило компьютерное зрение. За этим последовал прогресс в видео, графах и звуке, который привёл к сегодняшнему волнению вокруг больших языковых моделей, моделей, смешивающих текст и картинки, и моделей, которые генерируют.

С каждой волной менялся словарь: от коннекционизма 1980-х к глубокому обучению, которое отличило современные сети от меньших и более мелких моделей прошлого. Автор считает все эти ярлыки расплывчатыми, потому что современные искусственные сети почти не имеют отношения к настоящему мозгу. Что у них действительно общего, так это то, что они собраны из блоков, которые можно дифференцировать, блоков, чей отклик на крошечный сдвиг можно вычислить. Поэтому книга предпочитает более простое имя: дифференцируемые модели. С этой точки зрения нейронные сети принадлежат более широкой молодой дисциплине, которую называют дифференцируемым программированием: она смешивает информатику и оптимизацию, чтобы изучать любую компьютерную программу, которую можно настраивать таким способом.

Прогулка по этой стране — это ещё и прогулка по истории. Подгонка простых прямолинейных моделей повторными малыми поправками, метод четвёртой главы книги, была известна по крайней мере с девятнадцатого века. Полносвязные сети, которые используются дальше в книге, идут из 1980-х, а свёрточные модели для изображений уже применялись к концу 1990-х. Десятилетия ушли не на идеи, а на то, чтобы набралось достаточно данных и вычислительной мощности и стало видно, как хорошо эти модели работают, когда они достаточно велики. Телефон, который узнаёт ваше лицо, работает на идеях старше, чем вы могли бы предположить, и лишь недавно они стали практичными.



Старые идеи ждут мощности

Книга не может охватить всё, отчасти потому, что исследования движутся так быстро, но она стремится дать вам достаточно, чтобы вы сами находили дорогу в новейших статьях. Помимо печатных глав, автор обещает сайт с дополнительными главами о темах, которые не зависят от одного вида данных: генеративные модели, модели, которые включают и выключают части себя, повторное использование выученного одной моделью для другой задачи и объяснение того, что делают модели. Эти главы ближе к исследованиям, и читать их можно в любом порядке. Есть также практические занятия с подсказками в форме ноутбуков, которые следуют за большей частью книги и идут дальше, например в обучение сравнением пар примеров и в объединение моделей.

Запомним

Шумиха вокруг обучающихся машин не нова: уже в 1958 году перцептрон Розенблатта попал в заголовки. Названия вроде нейрона и слоя — исторические остатки, а волны интереса пришли с ImageNet в 2012 году и с трансформерами в 2017-м. Поскольку современные сети очень мало похожи на мозг, книга называет их дифференцируемыми моделями, частью более широкой области дифференцируемого программирования. Большинство ключевых идей родились десятилетия назад; не хватало лишь данных и вычислительной мощности, чтобы показать их силу.

Вопросы

1. Почему книга предпочитает название «дифференцируемые модели» названию «нейронные сети»?
 - a. Потому что современные сети копируют настоящие нейроны точнее, чем раньше
 - b. Потому что по-настоящему общее у современных сетей то, что они собраны из блоков, которые можно дифференцировать
 - c. Потому что слово «нейронный» запрещено в сегодняшних программах
 - d. Потому что дифференцируемые модели — совершенно новое изобретение 2017 года

2. Если ключевым идеям десятки лет, что так долго сдерживало эти модели?
 - a. Никто ещё не придумал идею поправлять модель шаг за шагом
 - b. Модели запретили после шумихи 1958 года
 - c. Не хватало данных и вычислительной мощности, чтобы показать, как хорошо работают большие модели
 - d. Полносвязные сети изобрели только в 2012 году
3. Почему книга всё же использует слова вроде нейронов, слоёв и активаций?
 - a. Чтобы совпадать с терминами существующих программ, хотя это исторические остатки
 - b. Потому что современные сети — точные модели мозга
 - c. Потому что у дифференцируемого программирования нет своих слов
 - d. Потому что конкурс ImageNet требовал этих названий

Ответы

1. **b)** Современные искусственные сети почти не имеют отношения к мозгу; их настоящая общая черта в том, что они собраны из дифференцируемых блоков.
2. **c)** Прямолинейная подгонка идёт из девятнадцатого века, полносвязные сети из 1980-х, а свёрточные модели из 1990-х; не хватало данных и вычислительной мощности.
3. **a)** Названия пришли из ранних попыток подражать нейронам мозга и остались в употреблении, поэтому книга сохраняет их для согласия с библиотеками, которыми все пользуются.

Задачи

1. Объясните своими словами, что книга имеет в виду под шуткой «шумиха вокруг ИИ, только на дворе 1958 год».

Подсказка: Подумайте, кем был Розенблатт и чем был перцептрон.

Ответ: В 1958 году перцептрон Фрэнка Розенблатта, одна из первых работающих нейронных сетей, привлёк большое внимание прессы, во многом как сегодняшнее волнение вокруг ИИ. Шутка напоминает, что воодушевление по поводу обучающихся машин приходит волнами и что основные идеи гораздо старше нынешнего бума.

2. Опишите две большие волны интереса к нейронным сетям, которые книга называет после 2012 года, и что изменила каждая из них.

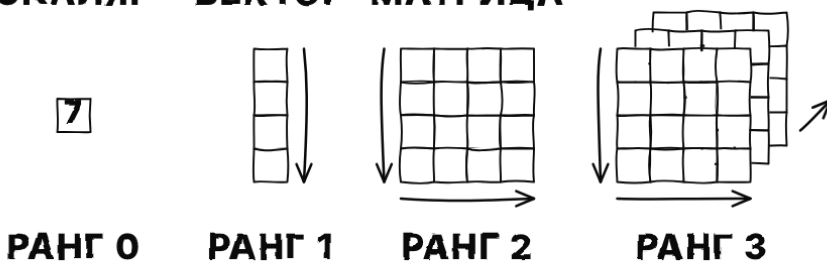
Подсказка: Одну волну двигало соревнование по фотографиям, другую — новый вид модели для языка.

Ответ: С 2012 по 2017 год большие соревнования, такие как ImageNet, заставили сети для распознавания изображений быстро расти в размере и сложности. С 2017 года трансформеры стремительно захватили обработку языка, как раньше свёрточные модели захватили зрение, а дальнейший прогресс в видео, графах и звуке привёл к сегодняшнему волнению вокруг больших языковых моделей и генеративных моделей.

Коробки с числами и их формы

Возьмите лист бумаги в клетку и запишите одно число в одну клетку. Потом запишите столбик чисел вниз, потом заполните целый прямоугольник, и наконец сложите несколько заполненных листов один на другой. Вы только что построили четыре вида объектов, из которых состоит вся книга. Одно число, столбик, сетка и стопка сеток различаются только одним: сколько направлений нужно, чтобы найти клетку.

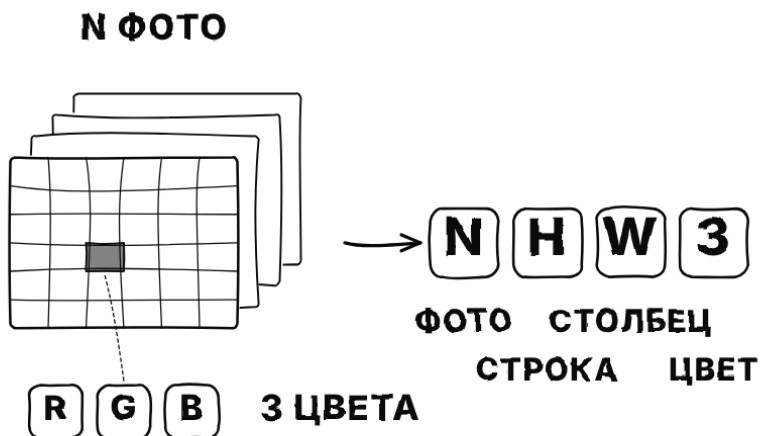
СКАЛЯР ВЕКТОР МАТРИЦА



Тензоры ранга от 0 до 3

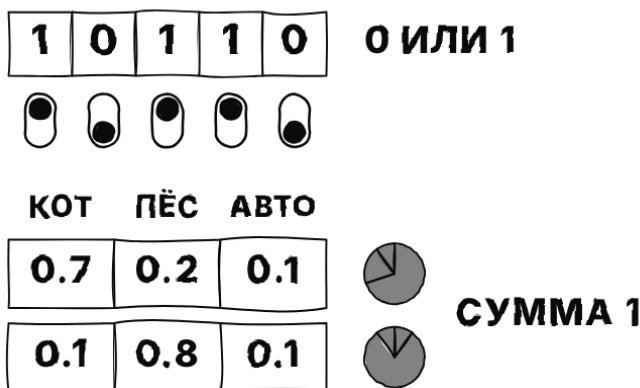
Книга называет их все тензорами, а число направлений называет рангом тензора. Одинокое число имеет ранг ноль и называется скаляром; столбик чисел имеет ранг один и называется вектором; прямоугольник чисел имеет ранг два и называется матрицей; у всего, где направлений больше, особого имени нет. Автор признаёт, что для математиков слово «тензор» значит нечто более строгое, и извиняется перед ними, но свободный смысл настолько распространён в программировании, что книга его сохраняет. Каждый вид получает в книге свой стиль буквы, чтобы с одного взгляда было видно, на что вы смотрите, а вычурные рукописные буквы отведены для коллекций и для вероятностей.

Теперь подумайте о фотографиях на вашем телефоне, загруженных в модель за один раз. Каждая фотография — сетка пикселей, каждый пиксель хранит три числа для красного, зелёного и синего, и фотографий сразу много. Так что весь пакет — это коробка с четырьмя направлениями: какая фотография, какая строка, какой столбец и какой из трёх цветов. Книга записывает такую коробку, перечисляя её размеры, форму, и оставляет часть из них буквами, потому что число фотографий или их высота могут меняться, а три цвета не меняются никогда. Главный совет автора: понять операцию очень часто сводится к тому, чтобы понять форму каждой коробки, которая входит и выходит.



Пакет фотографий как коробка

Тот же короткий знак, который объявляет форму, изредка используется и для извлечения случайного числа из распределения вероятностей; автор обещает, что смысл всегда будет ясен из контекста. Ещё две пометки говорят, что может лежать внутри коробки. Одна говорит, что в коробке только нули и единицы, как ряд выключателей, каждый из которых либо выключен, либо включён. Другая говорит, что числа никогда не отрицательны и в сумме дают единицу, как доли круговой диаграммы или как уверенность, которую приложение для фотографий распределяет между подписями «кошка», «собака» и «машина». Для больших коробок второго вида каждая строка сама по себе даёт в сумме единицу, так что у каждой фотографии своя круговая диаграмма.



$$1 \text{ ФОТО} = H \times W \times 3$$

Выключатели и диаграммы

Опыт

Откройте панель сведений любой фотографии на телефоне и найдите её размер, например ширину и высоту в пикселях. Перемножьте их в уме и добавьте, что каждый пиксель хранит три значения цвета: это размер коробки, которую модель видит для одной фотографии. Теперь представьте сотню таких фотографий, отправленных вместе, и у вас коробка с четырьмя направлениями, ровно такая, какую описывает книга.

Книга выросла из курса о нейронных сетях для науки о данных, который автор много лет читает в римском университете Сапиенца, с материалом, добавленным из других курсов, учебных пособий и летних школ. Она начинается на той же почве, что и другие хорошие недавние книги, а затем следует собственным исследовательским интересам автора, предлагая дополняющую точку зрения. Её главное убеждение в том, что эти модели понимаются через теорию и код одновременно, и автоматическое дифференцирование — лучший пример такого постоянного движения туда и обратно. Книга во многом приписывает возвращение нейронных сетей после 2012 года мощным программным библиотекам, от ранних, таких как Theano, Caffe и Chainer, до сегодняшних TensorFlow, PyTorch и JAX. Где только может, она связывает свои идеи с PyTorch и JAX, но это не руководство по программированию, и за этим она отправляет вас к документации каждой библиотеки.

Автор также честен в том, чем книга не является. Она выбирает идеи, распространённые сегодня и достаточно общие, чтобы остаться надолго, но будущего она не видит, и некоторые части могут устареть к тому времени, когда вы их прочитаете. Для каждой идеи она показывает несколько вариантов из исследовательских статей, например два разных способа держать числа в сети хорошо отмасштабированными, однако существуют сотни других, и исследовать их лучше на большом открытом сайте, который собирает статьи вместе с их кодом. Наконец, чтобы эти модели работали в большом масштабе, нужны инженерное мастерство и некоторая интуиция: об оборудовании книга говорит мало, а для интуиции ничто не заменит опыт и откровенные записи практиков в блогах.

Запомним

Всё в книге — тензор, коробка с числами, чей ранг равен числу направлений, нужных, чтобы найти один элемент: число, столбик, сетка или больше. Пакет цветных фотографий — коробка с четырьмя направлениями, и понять операцию обычно значит знать форму каждой участвующей коробки. Некоторые коробки хранят только нули и единицы, другие — неотрицательные числа, дающие в сумме единицу, как круговая диаграмма уверенности. Книга выросла из многолетнего университетского курса, соединяет теорию и код, опирается на PyTorch и JAX и признаёт, что остальное должны восполнить инженерия и опыт.

Вопросы

1. Что делает скаляр, вектор и матрицу разными видами тензора?
 - a. Величина чисел, которые они содержат
 - b. Сколько направлений нужно, чтобы найти один элемент
 - c. Целые это числа или с десятичными знаками
 - d. Взяты они из фотографий или из текста
2. Модель получает пакет цветных фотографий за один раз. Почему этот пакет — тензор с четырьмя направлениями?
 - a. Потому что у каждой фотографии четыре угла

- b. Потому что модели всегда нужно ровно четыре фотографии
 - c. Потому что нужно указать, какая фотография, какая строка, какой столбец и какой из трёх цветов
 - d. Потому что цветные фотографии хранятся как четыре отдельных файла
3. Приложение для фотографий сообщает, насколько оно уверено, что на картинке кошка, собака или машина. Какая пометка из книги подходит этим числам?
- a. Числа, которые никогда не отрицательны и в сумме дают единицу, как доли круговой диаграммы
 - b. Коробка, где только нули и единицы
 - c. Тензор ранга три без ограничений
 - d. Один скаляр без формы
4. По словам автора, что чаще всего открывает понимание операции в модели?
- a. Заучивание названий всех программных библиотек
 - b. Знание формы каждого тензора, который входит и выходит
 - c. Чтение характеристик оборудования
 - d. Подсчёт сносок в главе

Ответы

1. **b)** Число, столбик, сетка и стопка сеток различаются только тем, сколько направлений нужно, чтобы найти клетку, и это число есть ранг.

2. c) Каждая фотография — сетка пикселей с тремя значениями цвета на пиксель, и фотографий приходит много сразу, так что для поиска одного числа нужны четыре выбора.

3. a) Уверенности, распределённые по подписям, неотрицательны и дают в сумме единицу, как доли круговой диаграммы; для многих фотографий каждая строка получает свою диаграмму.

4. b) Главный совет книги: понять операцию обычно сводится к тому, чтобы понять форму каждого участвующего тензора.

Задачи

1. Модель читает пакет коротких звуковых клипов. Каждый клип — длинная строка значений громкости, и у всех клипов в пакете одинаковая длина. Сколько направлений у пакета и что означает каждое направление?

Подсказка: Спросите себя, что нужно выбрать, чтобы указать на одно значение громкости.

Ответ: Два направления: какой клип и какой момент внутри клипа. Чтобы найти одно значение, нужны оба выбора, так что пакет — это сетка чисел, тензор ранга два, матрица.

2. Объясните, почему книга оставляет некоторые размеры в форме буквами, а другие записывает фиксированными числами, на примере пакета цветных фотографий.

Подсказка: Какие размеры могут меняться от пакета к пакету, а какие не меняются никогда?

Ответ: Число фотографий, их высота и ширина могут отличаться от одного применения к другому, поэтому их оставляют буквами; у каждого пикселя всегда ровно три значения цвета, поэтому этот размер записан фиксированным числом.

3. Скажите своими словами, чем книга, по её собственному признанию, не является и куда она отправляет читателя за этим.

Подсказка: Подумайте о подробностях программирования, полноте и о том, как заставить модели работать в большом масштабе.

Ответ: Это не руководство по программированию, поэтому она указывает на документацию PyTorch и JAX. Она не полна и не защищена от устаревания, поэтому за многими вариантами каждой идеи отправляет на открытый сайт, собирающий статьи с их кодом. И она мало говорит об оборудовании и о ремесле запуска моделей в большом масштабе, где лучшие учителя — опыт и записи практиков в блогах.

Стрелки, которые смотрят в одну сторону

Возьмите два карандаша и держите их перед собой так, чтобы оба начинались от одного кончика пальца. Направьте их ровно в одну сторону, потом раскройте под прямым углом, потом поверните один, пока они не будут смотреть в противоположные стороны. Без всяких чисел вы уже можете сказать, насколько два карандаша согласны: полностью, никак или совсем наоборот. К концу этой части вы увидите, что одна простая операция измеряет ровно это согласие и что она лежит в сердце почти каждой модели в книге.



СОГЛАСНЫ



НИКАК



ПРОТИВ

Согласие, безразличие, спор

Сначала пара слов о самой книге, из её благодарностей. Цветные формулы сделаны свободным пакетом вёрстки другого автора, цветные картинки Алисы и маленькие значки на полях лицензированы у фотобанка, а рисунки Алисы внутри глав — оригинальные иллюстрации Джона Тенниела, взятые с Викисклада. Автор благодарит коллег и читателей, которые присылали исправления и подталкивали его к публикации.

Лицензия важна для того, что вы слушаете сейчас. Книга выпущена под лицензией Creative Commons, которая позволяет любому делиться ею, переделывать, адаптировать и строить на ней что угодно, в любой форме и даже коммерчески, при условии указания автора. Есть одно условие: всё, что построено на ней,

должно выходить на тех же условиях. Этот курс — ровно такая переделка, поэтому он несёт ту же лицензию и не использует лицензированные картинки из фотобанка, на которые лицензия книги не распространяется.

Затем вторая глава собирает математику, которая понадобится остальной книге. Автор предполагает, что вы уже встречали эти темы, и использует главу главным образом для того, чтобы договориться об обозначениях и дать связный обзор, указывая, где идея встречается с настоящим кодом. Она движется в три шага, каждый опирается на предыдущий: линейная алгебра, затем градиенты для коробок чисел любого ранга, затем то, как градиенты используют, чтобы улучшать функцию. Вероятности отведено своё приложение, построенное вокруг принципа максимального правдоподобия, и автор по-доброму предупреждает, что глава плотно набита определениями.

Формально тензор — это коробка элементов одного типа, и описывается он этим типом и своей формой, списком размеров, который мы уже встречали. Тензоры так важны потому, что они подходят массово параллельному оборудованию: графические чипы игровых компьютеров, а тем более специализированные чипы, могут работать с тысячами элементов одновременно. Когда чат-ассистент отвечает вам за секунды, он прогоняет через такие чипы огромные коробки чисел. Большинство коробок в книге хранят числа с десятичными знаками, но в них могут быть и целые числа, например метки классов, или даже куски текста. Из коробки можно вырезать ломтики, строку, блок, один элемент, по правилам популярной библиотеки NumPy, только книга считает с единицы, а не с нуля.

+	10	20	30	40
1	11	21	31	41
2	12	22	32	42
3	13	23	33	43
4	14	24	34	44

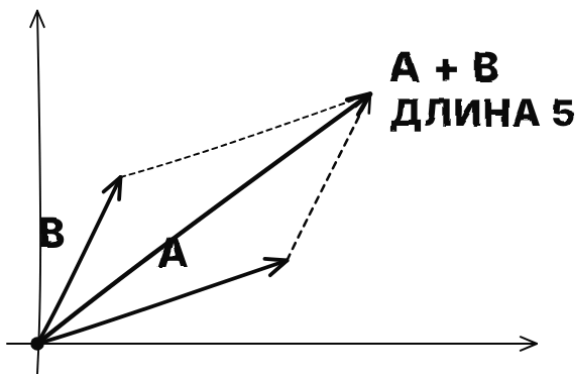
16, А НЕ 4

Четыре плюс четыре — шестнадцать

Почти каждая модель в книге — цепочка самых обычных операций: суммы, умножения и несколько изгибающих функций, таких как экспонента, синус, косинус и квадратный корень. Учебники тщательно различают вектор, записанный столбцом, и тот же вектор, записанный строкой. В коде это различие хитрее, потому что столбец, строка и простой список чисел — три разные формы. Библиотеки следуют правилу, которое называют **broadcasting**: когда две коробки не совпадают, меньшую тихо повторяют, пока они не совпадут. Книга показывает классическую аварию: вы прибавляете столбик из четырёх чисел к простому списку из четырёх чисел, ожидая четыре результата, а получаете сетку из шестнадцати, и ни одного сообщения об ошибке.

Опыт

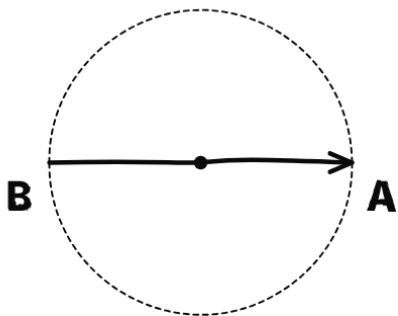
Нарисуйте на листе бумаги маленький крестик для начала координат и проведите от него две стрелки. Теперь сдвиньте копию второй стрелки так, чтобы она начиналась на острие первой, и проведите стрелку от начала координат туда, где вы оказались. Вы только что сложили два вектора: результат — диагональ параллелограмма, который образуют две стрелки. Измерьте результат линейкой: эта прямая длина от начала координат и есть то, что книга называет нормой.



Сложение стрелок

Такова геометрическая картина за векторами. Любые два вектора одной формы можно растянуть и сложить, элемент к элементу, и получить третий, и эту свободу математики называют векторным пространством. Вектор, если видеть в нём точку в пространстве, имеет длину, расстояние от начала координат, и книга часто использует квадрат длины, а это просто все элементы, возведённые в квадрат и сложенные.

Самая важная операция над векторами во всей книге — скалярное произведение, его ещё называют внутренним произведением. Вы перемножаете два вектора элемент к элементу и складываете всё, что получилось, так что результат — одно число, а не вектор. Маленький пример в книге перемножает два коротких списка из трёх чисел и заканчивается одним слегка отрицательным числом. Его смысл геометрический: скалярное произведение — это две длины, перемноженные между собой и умноженные на меру того, насколько хорошо выровнены две стрелки. Когда у обеих стрелок длина единица, остаётся только само выравнивание, которое называют косинусным сходством. Оно наибольшее, когда стрелки смотрят в одну сторону, наименьшее, когда в противоположные, и ровно ноль, когда они стоят под прямым углом, что книга называет ортогональностью. Функции поиска, которые находят фотографии или сообщения, близкие по смыслу к вашему запросу, обычно превращают каждый элемент в такую стрелку и сравнивают направления именно так.



$$\mathbf{A} \cdot \mathbf{B} = \sum A_i B_i$$

$$\cos = -1.00$$

Косинусное сходство

Запомним

Благодарности в книге упоминают пакет для формул, лицензированные картинки из фотобанка и оригинальные рисунки Алисы Тенниела. Её лицензия позволяет любому переделывать её, даже коммерчески, с указанием автора и на тех же условиях, поэтому этот курс существует и поэтому он избегает картинок из фотобанка. Вторая глава повторяет линейную алгебру, градиенты и оптимизацию; тензоры — коробки одного типа с формой, идеальные для параллельных чипов, а broadcasting может молча превратить четыре результата в шестнадцать. Скалярное произведение перемножает два вектора элемент к элементу и складывает, давая одно число, которое измеряет длины и выравнивание вместе. Для стрелок длины единица это косинусное сходство: наибольшее для одного направления, наименьшее для противоположных, ноль для прямого угла.

Вопросы

1. Две стрелки длины единица стоят под прямым углом друг к другу. Каково их косинусное сходство?
 - a. Наибольшее, потому что у обеих стрелок длина единица
 - b. Наименьшее, потому что они смотрят в разные стороны
 - c. Ровно ноль, потому что они ортогональны
 - d. Этого нельзя сказать, не зная каждый элемент

2. Вы прибавляете столбик из четырёх чисел к простому списку из четырёх чисел и получаете сетку из шестнадцати чисел. Что произошло?
- a. Библиотека сделала арифметическую ошибку
 - b. Бродкастинг тихо повторил коробки, чтобы их формы совпали
 - c. Вместо суммы было вычислено скалярное произведение
 - d. Числа были слишком большими и переполнились
3. Почему тензоры так важны в глубоком обучении?
- a. Потому что они подходят массово параллельным чипам, которые работают с тысячами элементов сразу
 - b. Потому что они могут хранить только числа с десятичными знаками
 - c. Потому что они превращают каждый вектор в столбец
 - d. Потому что они избавляют от необходимости в градиентах
4. Что измеряет скалярное произведение двух векторов в геометрическом прочтении книги?
- a. Только разницу их длин
 - b. Число общих у них элементов
 - c. Их две длины, перемноженные между собой и умноженные на то, насколько хорошо выровнены стрелки
 - d. Площадь параллелограмма, который они образуют

Ответы

1. **с)** Косинусное сходство наибольшее для одного направления, наименьшее для противоположных и равно ноль для прямого угла, что книга называет ортогональностью.
2. **б)** Столбик и простой список — разные формы, и бродкастинг молча повторяет меньшую коробку, пока формы не совпадут, давая сетку вместо четырёх результатов.
3. **а)** Коробки чисел одного типа идеальны для графических чипов и другого специализированного оборудования, которое обрабатывает много элементов одновременно.
4. **с)** Скалярное произведение равно произведению длин на меру выравнивания; для стрелок длины единица остаётся только выравнивание, косинусное сходство.

Разобранные примеры

Скалярное произведение из книги Книга берёт два коротких вектора: первый с элементами 0,1, 0 и минус 0,3, второй с элементами минус 4,0, 0,05 и 0,1. Каково их скалярное произведение?

Перемножим первые элементы: 0,1 умножить на минус 4,0 даёт минус 0,4.

Перемножим вторые элементы: 0 умножить на 0,05 даёт 0.

Перемножим третьи элементы: минус 0,3 умножить на 0,1 даёт минус 0,03.

Сложим три результата, потому что скалярное произведение — сумма поэлементных произведений: минус 0,4 плюс 0 минус 0,03.

Ответ: Минус 0,43, одно число, слегка отрицательное, так что две стрелки смотрят несколько против друг друга.

Задачи

1. Ничего не вычисляя, скажите, положительно, отрицательно или равно нулю скалярное произведение двух векторов, когда две стрелки смотрят ровно в противоположные стороны, и почему.

Подсказка: Вспомните, что делает мера выравнивания для противоположных направлений.

Ответ: Отрицательно. Скалярное произведение — это две длины, которые положительны, умноженные на выравнивание, а для противоположных направлений выравнивание наименьшее, отрицательное значение.

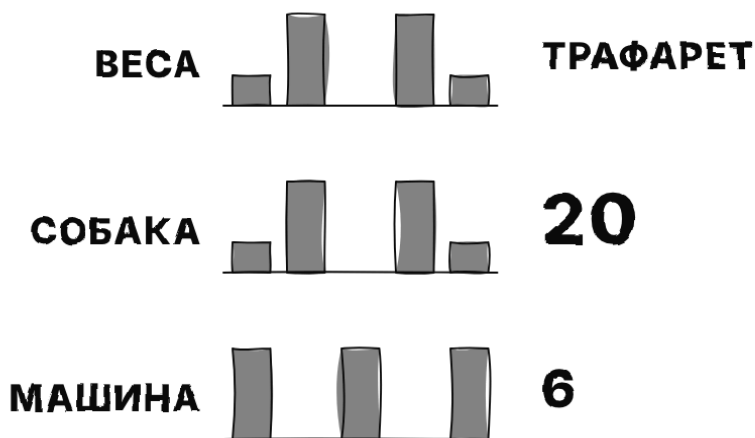
2. Объясните своими словами, что разрешает лицензия книги и какое одно условие к ней прилагается.

Подсказка: Подумайте о том, чтобы делиться, менять, продавать, и о том, что должно произойти с новой работой.

Ответ: Любой может делиться книгой, переделывать, адаптировать её и строить на ней, в любой форме и даже коммерчески, при условии указания автора. Всё, что построено на ней, должно выходить под той же лицензией, поэтому этот курс тоже её несёт; лицензированные картинки из фотобанка в это разрешение не входят и не используются.

Трафареты, пакеты и машины из чисел

Возьмите трафарет, лист картона с вырезанной фигурой, и поводите им над страницей с напечатанными фигурами. Где фигура на странице точно совпадает с отверстием, она видна полностью; где совпадает лишь наполовину, часть её скрыта. Скалярное произведение ведёт себя ровно как этот трафарет. Закрепите одну стрелку длины единица и примеряйте к ней все остальные стрелки длины единица: скалярное произведение наибольшее ровно тогда, когда вторая стрелка совпадает с первой.



Веса как трафарет

На этом простом факте держится большая часть книги. В следующих главах одна стрелка будет входом, скажем фотографией, а другая — подстраиваемыми параметрами модели, которые книга называет весами. Скалярное произведение наибольшее, когда вход резонирует с весами, так что веса работают как трафарет, к которому прикладывают вход, и книга называет это сопоставлением с шаблоном. Можно представить, что приложение для фотографий держит трафарет для собак: фотография, которая хорошо с ним совпадает, даёт большое число, а фотография машины — маленькое.

Попутно два маленьких наблюдения. Сложить все элементы вектора — то же самое, что взять его скалярное произведение с

вектором, состоящим целиком из единиц. А расстояние между двумя векторами можно записать через одни лишь скалярные произведения; в частности, скалярное произведение вектора с самим собой — это квадрат его длины. Оба приёма встречаются снова и снова, на бумаге и в коде.

Дальше идут матрицы, сетки с некоторым числом строк и столбцов. Книга просит видеть в матрице стопку векторов, по одному на строку, и называет такую стопку пакетом данных. Представьте таблицу, где каждая строка — одна фотография, описанная одним и тем же списком чисел: весь лист — одна матрица, и модели пишут так, чтобы они работали сразу с целыми такими листами.

1	2	0
0	1	3
2	2	1

×

1
0
2

=

1
6
4

$1 + 0 + 0 = 1$

$0 + 0 + 6 = 6$

$2 + 0 + 2 = 4$

ВСЁ СРАЗУ

Строка на столбец

Ключевая операция над матрицами — умножение матриц, и построено оно из скалярного произведения. Каждый элемент результата — скалярное произведение одной строки первой матрицы с одним столбцом второй, поэтому строки первой должны быть ровно такой же длины, как столбцы второй. Когда вторая матрица — один столбец, получается частный случай: матрица, умноженная на вектор. Вот практическая выгода: если первая матрица — пакет входов, а вторая хранит веса, одно единственное умножение вычисляет скалярное произведение каждого входа с весами, всё сразу. А пакет, умноженный на свою перевернутую версию, даёт за один раз скалярное произведение каждой пары строк, полную таблицу того, насколько каждый элемент похож на каждый другой.

Опыт

Составьте маленькую таблицу из трёх строк, каждая — короткий список из трёх чисел, и четвёртый список из трёх чисел в отдельном столбце как ваши веса. Рядом с каждой строкой попросите таблицу вычислить сумму произведений соответствующих элементов на веса. Вы вычислили вручную три скалярных произведения, по одному на строку; умножение матриц делает ровно это, для тысяч строк и многих столбцов весов за один шаг.

Матрицы можно умножать и поэлементно, когда у них одна форма, по правилу, названному в честь математика Адамара. Ему не хватает изящных свойств настоящего умножения матриц, но оно чрезвычайно удобно: умножение на сетку из нулей и единиц выключает элементы, как маска, а умножение на другие числа

масштабирует их. Некоторые недавние семейства моделей тоже опираются на такие поэлементные произведения, как покажут следующие главы.

$$\begin{array}{|c|c|c|} \hline 5 & 2 & 7 \\ \hline 1 & 8 & 3 \\ \hline 4 & 6 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 5 & 0 & 7 \\ \hline 0 & 8 & 0 \\ \hline 4 & 6 & 0 \\ \hline \end{array}$$

МАСКА

Маска, элемент за элементом

В том же духе, когда книга применяет функцию, например экспоненту, к целой матрице, имеется в виду применить её к каждому элементу отдельно. Это не то же самое, что настоящая матричная экспонента из учебников, которая работает только с квадратными матрицами и строится из повторных умножений матриц. Поэлементная версия работает с коробками любой формы, поэтому библиотеки вроде PyTorch держат по-настоящему матричные операции, такие как обращение и определители, в отдельном уголке своего набора инструментов.

Блок кода в книге показывает обе рядом: один обычный вызов для поэлементной экспоненты и один специальный для матричной экспоненты. Наконец, редукция сжимает коробку вдоль одного направления, например суммируя или усредняя каждую строку, и в коде её выбирают, называя направление.



МАШИНЫ, СОЕДИНЁННЫЕ ВМЕСТЕ

Матрицы — это машины

Заключительный блок спрашивает, почему умножение матриц определено именно так, строка на столбец, а не поэлементно. Ответ в том, что оно схватывает линейные функции, функции, которые уважают растяжение и сложение: растяните или сложите входы сначала или выходы потом, и вы получите тот же результат. Каждую такую функцию можно записать как матрицу. Умножить матрицу на вектор — значит просто применить

функцию, а умножить две матрицы — выполнить одну функцию за другой. Так что матрица — не просто таблица чисел; это маленькая машина, а умножение матриц соединяет машины вместе.

Запомним

Среди стрелок длины единица скалярное произведение наибольшее, когда обе одинаковы, так что веса модели работают как трафарет, с которым резонирует вход: сопоставление с шаблоном. Матрицу можно видеть как пакет, стопку векторов, по одному на строку. Умножение матриц заполняет каждый элемент скалярным произведением строки и столбца, так что одно умножение вычисляет все скалярные произведения пакета сразу. Поэлементное умножение маскирует и масштабирует, функции вроде экспоненты действуют поэлементно, а редукции сжимают коробку вдоль одного направления. Умножение матриц определено так потому, что матрица — линейная функция: умножение на вектор применяет её, а умножение матриц соединяет функции в цепочку.

Вопросы

1. Почему книга называет скалярное произведение входа и весов «сопоставлением с шаблоном»?
 - а. Потому что оно копирует вход в веса

- b. Потому что оно наибольшее, когда вход совпадает с весами, как фигура, подходящая к трафарету
 - c. Потому что оно считает, сколько элементов у двух векторов
 - d. Потому что для фотографий машин оно всегда даёт ноль
2. Пакет входов, по одному на строку, умножают на матрицу с весами. Чего достигает это одно умножение?
- a. Оно усредняет все входы в одну строку
 - b. Оно вычисляет скалярное произведение каждого входа с весами, всё сразу
 - c. Оно умножает входы на веса поэлементно
 - d. Оно возводит каждый элемент пакета в квадрат
3. Вы хотите выключить некоторые элементы матрицы, а остальные оставить без изменений. Какая операция подходит?
- a. Поэлементное умножение на сетку из нулей и единиц той же формы
 - b. Настоящая матричная экспонента
 - c. Умножение матрицы на её перевёрнутую версию
 - d. Редукция, которая суммирует каждую строку
4. Почему умножение матриц определено как строка на столбец, а не поэлементно?
- a. Потому что так быстрее считать на графических чипах
 - b. Потому что поэлементно можно умножать только квадратные матрицы

- c. Потому что матрица представляет линейную функцию, и это правило делает умножение матриц соединением функций в цепочку
- d. Потому что поэлементное умножение не разрешено в PyTorch

Ответы

- 1. **b)** Среди стрелок длины единица скалярное произведение наибольшее, когда обе одинаковы, так что веса работают как трафарет, с которым резонирует вход.
- 2. **b)** Каждый элемент произведения матриц — скалярное произведение строки со столбцом, так что одно умножение даёт все скалярные произведения пакета вместе.
- 3. **a)** Умножение Адамара, то есть поэлементное, на нули и единицы работает как маска, выключая элементы там, где в маске нули.
- 4. **c)** Каждую линейную функцию можно записать как матрицу; умножение на вектор применяет её, а умножение двух матриц выполняет одну функцию за другой.

Разобранные примеры

Какие формы можно перемножить В пакете 3 строки по 4 числа в каждой, а в матрице весов 4 строки и 2 столбца. Можно ли их перемножить и какой формы будет результат?

Каждый элемент произведения — скалярное произведение строки первой матрицы со столбцом второй, так что они должны быть одной длины.

В строке пакета 4 числа, и в столбце весов 4 числа, так что умножение возможно.

На каждую пару из строки пакета и столбца весов приходится один элемент результата: 3 строки в паре с 2 столбцами.

Ответ: Да; результат имеет 3 строки и 2 столбца, по одной строке на вход и по одному столбцу на набор весов.

Задачи

1. Без формулы объясните, почему скалярное произведение вектора с вектором, состоящим целиком из единиц, даёт сумму его элементов.

Подсказка: Подумайте, что делает умножение каждого элемента на единицу перед тем, как всё сложат.

Ответ: Скалярное произведение перемножает соответствующие элементы и складывает результаты. Умножение каждого элемента на единицу оставляет его без изменений, так что складываются сами элементы, а это и есть их сумма.

2. Пакет фотографий, по одной на строку, умножают на его собственную перевёрнутую версию. Опишите таблицу, которую вы получите, и что говорит один её элемент.

Подсказка: Каждый элемент соединяет в пару одну строку с одной строкой.

Ответ: Вы получаете квадратную таблицу с одной строкой и одним столбцом на фотографию. Каждый элемент — скалярное произведение строк двух фотографий, мера того, насколько похожи эти два элемента, так что вся таблица сравнивает каждый элемент с каждым другим за один шаг.

3. Объясните разницу между применением экспоненты к матрице поэлементно и настоящей матричной экспонентой, и почему библиотеки держат их отдельно.

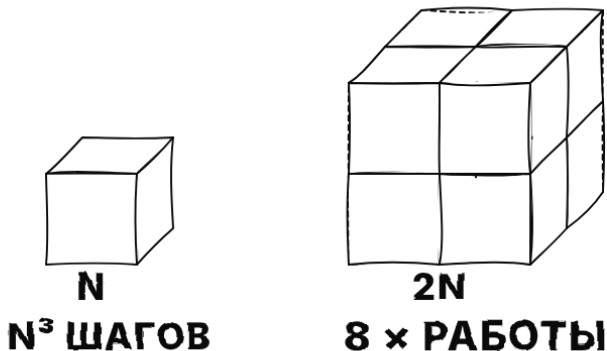
Подсказка: Спросите себя, какая из них работает с коробками любой формы.

Ответ: Поэлементно экспонента применяется к каждому числу отдельно и работает с коробками любой формы. Настоящая матричная экспонента строится из повторных умножений матриц и работает только для квадратных матриц. Поскольку чисто матричные операции, такие как эта, обращение и определители, особенные, библиотеки вроде PyTorch держат их в отдельном модуле.

Сколько работы в одном умножении

Представьте, что вы вручную заполняете большую таблицу умножения, где каждая клетка требует сопоставить целую строку одной матрицы с целым столбцом другой. Теперь сделайте обе матрицы вдвое шире и вдвое выше и заполните новую таблицу. Вы ожидали бы вдвое больше работы, но обнаружите, что работаете примерно в восемь раз дольше. Эта неожиданность и есть то, что информатики называют сложностью, и книга вводит её на примере умножения матриц.

$$O(N^3)$$



Вдвое больше, в восемь раз дольше

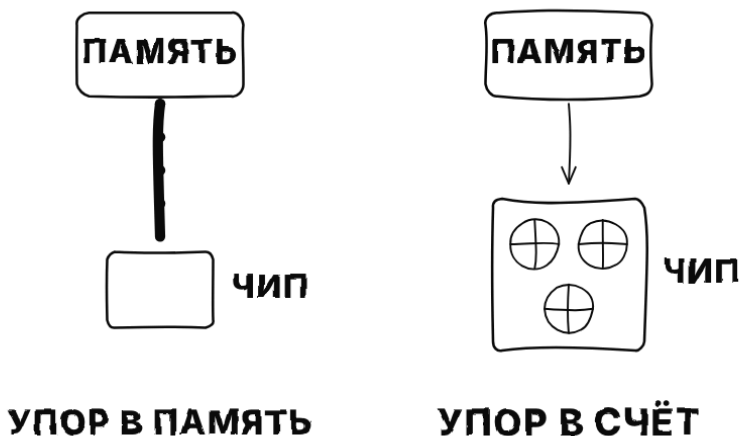
Чтобы перемножить две матрицы напрямую по определению, вы вычисляете одно скалярное произведение на каждую клетку результата, и каждое из них по длине равно общей стороне двух матриц. Общее число шагов называют временной сложностью. Память, которая нужна, если вычислять клетки одну за другой, — это просто размер результата, и это называют пространственной сложностью.

Чтобы говорить о стоимости, не привязываясь к конкретному компьютеру, информатика использует так называемую нотацию O большое, от немецкого слова «порядок». Она описывает, как растёт стоимость с ростом задачи, и намеренно игнорирует постоянные множители, которые перестают иметь значение, когда задача достаточно велика; такой способ рассуждения называют асимптотическим анализом. В этих терминах простой способ умножения матриц растёт пропорционально каждому из трёх участвующих размеров. Для двух квадратных матриц он растёт как куб их размера, и ровно поэтому удвоение размера означало восьмикратную работу.

Книга предупреждает, что эта изящная картина не рассказывает всей истории. Известны более хитрые методы умножения квадратных матриц, которые растут заметно медленнее куба, однако их трудно распределить по тысячам маленьких работников внутри графического чипа, поэтому на практике ими почти никто не пользуется. Что на самом деле происходит, когда вы просите библиотеку вроде PyTorch перемножить две матрицы, зависит ещё и от машины: библиотека выбирает низкоуровневую процедуру, написанную для этого оборудования, одну для центрального процессора, другую для графического чипа, третью для специализированных чипов. Компиляторы могут пойти

дальше и слить несколько операций в одну, настроенную под конкретный чип. Один и тот же чат-ассистент может работать на телефоне или в дата-центре, и внизу умножением занимаются совсем разные процедуры.

Стоит запомнить ещё два предела теории. Много процессоров, работающих параллельно, лишь делят работу на постоянный множитель, так что это не меняет того, как стоимость растёт с размером. И теория игнорирует время, которое уходит на перемещение данных из одного места памяти в другое, а в настоящих программах именно это часто и есть истинное узкое место. Когда программу сдерживает перемещение данных, говорят, что она упирается в память; когда предел — сама арифметика, говорят, что она упирается в вычисления. Единственный надёжный способ понять, что из двух происходит, — измерить работающую программу профилировщиком, и книга обещает, что взаимодействие теории и измерения будет возвращаться снова и снова.



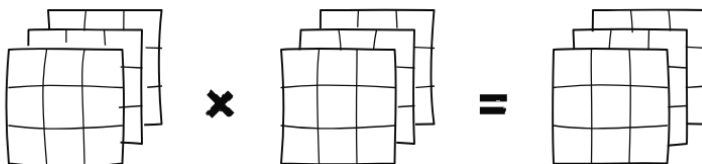
Упор в память или в вычисления

Опыт

Представьте кухню с одним поваром и холодильником в соседней комнате. Если каждое блюдо требует одного быстрого взмаха ножом, но похода к холодильнику за каждым ингредиентом, нанять ещё поваров почти не поможет: узкое место — ходьба, и кухня упирается в память. Если же каждое блюдо требует часов нарезки, а всё уже лежит на столе, лишние руки помогают напрямую: кухня упирается в вычисления.

Наконец, книга смотрит дальше векторов и матриц. Многие особые операции, такие как определители или матричная экспонента, имеют смысл только для матриц, поэтому большинство операций над большими коробками — это либо много матричных операций, выполненных бок о бок, либо сочетания матричных операций с редукциями. Первый вид — пакетное умножение матриц: взять стопку матриц и вторую стопку той же длины и перемножить их пара за парой, получив стопку результатов. Библиотеки без просьбы считают первые направления коробки пакетными, так что в PyTorch пакетная версия записывается ровно так же, как обычное умножение матриц, как показывает блок кода в книге. Когда приложение на телефоне обрабатывает много фотографий сразу, оно опирается именно на эту привычку обрабатывать целые стопки за один вызов.

ПАКЕТ ИЗ 3



ВЫТЯНУТЬ, ЗАТЕМ •

Стопки, перемноженные попарно

Второй вид — обобщённое скалярное произведение: две коробки одной формы, с любым числом направлений, перемножаются поэлементно, а затем всё суммируется. Это не более чем вытянуть каждую коробку в одну длинную строку чисел и взять обычное скалярное произведение. С этими операциями, говорит книга, она охватила большинство операций над тензорами, нужных для остальной книги.

Запомним

Прямое умножение матриц стоит одно скалярное произведение на элемент результата, это временная сложность, и требует памяти под результат, это пространственная сложность. Нотация O большое описывает, как стоимость растёт с размером, и для квадратных матриц простой метод растёт как куб размера, так что удвоение размера означает примерно восьмикратную работу. Более быстрые методы существуют на бумаге, но плохо подходят параллельным чипам, библиотеки выбирают процедуры под конкретное оборудование, а много процессоров лишь делят стоимость на постоянный множитель. Перемещение данных часто и есть настоящее узкое место, в этом разница между программами, упирающимися в память и в вычисления, и лишь измерение показывает, какой случай перед вами. Большие коробки обрабатываются как стопки матричных операций бок о бок или как обобщённые скалярные произведения вытянутых коробок.

Вопросы

1. Две квадратные матрицы делают вдвое больше по каждой стороне. Как примерно меняется работа простого умножения?
 - а. Удваивается

- b. Растёт примерно в четыре раза
 - c. Растёт примерно в восемь раз
 - d. На графическом чипе остаётся той же
2. Почему более хитрые методы умножения матриц, растущие медленнее, редко используются на практике?
- a. Они дают неверные ответы для больших матриц
 - b. Их трудно распределить по множеству маленьких работников графического чипа
 - c. Им нужны неквадратные матрицы
 - d. Их запрещает нотация O большое
3. Программа едва ускоряется, когда вы добавляете процессоры, а измерение показывает, что большую часть времени она тратит на выборку данных. Что это?
- a. Упор в вычисления
 - b. Упор в память
 - c. Асимптотически кубическая
 - d. Свободная от пространственной сложности
4. Что нотация O большое намеренно опускает?
- a. То, как стоимость растёт с ростом задачи
 - b. Постоянные множители, которые перестают иметь значение, когда задача достаточно велика
 - c. Размер входов
 - d. То, занимает ли операция время вообще

Ответы

1. **с)** Для квадратных матриц простой метод растёт как куб размера, так что удвоение размера означает примерно восьмикратную работу.
2. **б)** Хотя на бумаге они растут медленнее куба, они плохо распараллеливаются на сильно параллельном оборудовании, таком как графические чипы.
3. **б)** Когда программу сдерживает перемещение данных, а не арифметика, она упирается в память, и надёжно показать это может только профилировщик.
4. **б)** О большое описывает, как стоимость растёт с размером, и игнорирует постоянные множители, поэтому много параллельных процессоров её не меняют.

Разобранные примеры

Пакетное умножение из книги В блоке кода книги первая стопка содержит 4 матрицы из 5 строк и 2 столбцов, а вторая — 4 матрицы из 2 строк и 3 столбцов. Какая форма получается из пакетного умножения?

Первое направление — пакет: есть 4 пары, перемножаемые пара за парой, так что результат — стопка из 4 матриц.

В каждой паре матрица 5 на 2 встречается с матрицей 2 на 3; общая сторона 2 совпадает, так что их можно перемножить.

У каждого произведения одна строка на строку первой матрицы и один столбец на столбец второй: 5 строк и 3 столбца.

Ответ: Стопка из 4 матриц, каждая из 5 строк и 3 столбцов.

Задачи

1. Объясните, почему много процессоров не меняют сложность операции в смысле O большое, даже если программа становится быстрее.

Подсказка: Какое ускорение в лучшем случае дают много процессоров?

Ответ: Много процессоров в лучшем случае делят работу на постоянный множитель, число процессоров. Оно большое игнорирует постоянные множители и описывает только то, как стоимость растёт с размером, так что рост остаётся тем же, хотя каждый запуск заканчивается раньше.

2. Опишите своими словами обобщённое скалярное произведение двух коробок с тремя направлениями и скажите, как оно связано с обычным скалярным произведением.

Подсказка: Представьте, что каждую коробку разворачивают в одну длинную строку.

Ответ: Две коробки одной формы перемножаются поэлементно, и все произведения складываются в одно число. Это ровно обычное скалярное произведение двух коробок после того, как каждую вытянули в одну длинную строку чисел.

Рецепт из букв и наклон кривой

Представьте, что вы даёте каждому направлению коробки чисел букву, как подписываете ящики шкафа: одна буква для того, какая фотография, одна для того, какая строка, одна для того, какой цвет. Теперь опишите операцию, просто перечислив буквы того, что входит, и буквы того, что должно выйти. Любая буква, которая по пути исчезла, была просуммирована. Эта крошечная идея, заимствованная из сокращённой записи, которой Альберт Эйнштейн пользовался в физике, называется `einsum`, и книга преподносит её как необязательный, но очень удобный раздел.

Возьмём две операции из прошлой части. Пакетное умножение матриц становится короткой строкой, которая говорит: входят две стопки с общей буквой посередине, а выходит стопка без этой буквы; общая буква и есть та, что суммируется, ровно скалярные произведения строк и столбцов. Обобщённое скалярное произведение становится ещё более короткой строкой: входят две коробки с одними и теми же тремя буквами, а не выходит ничего, так что всё суммируется в одно число. Между этими строками и полными письменными определениями есть прямое соответствие, и большинство библиотек предлагают функцию с именем `einsum`, которая их читает.

Преимущество практическое. Вам больше не нужно помнить, у какой из нескольких функций умножения в библиотеке какие правила для форм, потому что одна строка покрывает их все. Одна и та же строка без изменений работает в PyTorch и в JAX, так что перенос кода из одной библиотеки в другую становится

прозрачным. Перевернуть матрицу внутри операции так же просто, как поменять две буквы в строке местами. По этим причинам `einsum` и инструменты, которые его расширяют, например популярный пакет `einops`, стали широко использоваться.

Теперь книга обращается к сердцу слова «дифференцируемый»: к градиентам. Именно они позволяют моделям улучшать себя почти автоматически, методом, который называют градиентным спуском, и книга напоминает самое главное, предпочитая пользу полной математической строгости. Она начинается с простейшего случая: функция, которая берёт одно число и даёт одно число.

Опыт

Нарисуйте гладкую кривую, которая опускается, а потом снова поднимается, как пологая долина, и выберите точку на её склоне. Положите линейку так, чтобы она лишь касалась кривой в этой точке, не пересекая её. Теперь сдвиньтесь по кривой на крошечный шаг вправо: если линейка наклонена вверх, кривая там поднимается; если наклонена вниз, кривая падает. Наклон линейки и есть производная в этой точке.



Наклон касательной

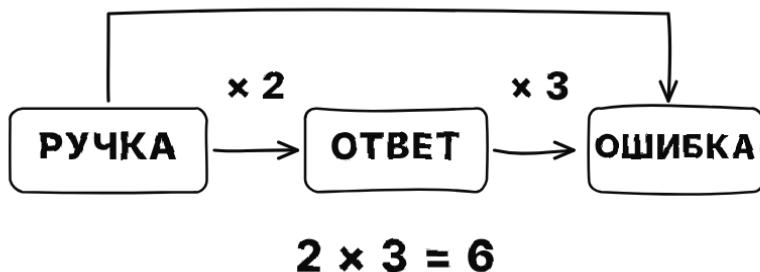
Ровно такова картина в книге. Производная определяется вопросом: насколько меняется выход при крошечном изменении входа, когда это крошечное изменение стягивается к нулю. Геометрически это наклон прямой, касающейся кривой в точке, а также лучшее прямолинейное подражание функции вблизи этой точки. Рисунок в книге показывает пологую кривую в форме долины с двумя такими касательными в двух разных точках, одна наклонена вниз, другая вверх. Положительный наклон означает, что в маленькой окрестности функция идёт вверх направо и вниз налево, а отрицательный наклон означает обратное. Это как датчик наклона в вашем телефоне: он говорит, в какую сторону наклонена земля ровно там, где вы стоите, а не что лежит в миле

от вас. Книга подчёркивает, что этот локальный взгляд без изменений переносится на функции многих чисел.

Знакомые школьные правила для степеней, логарифмов и синусов все напрямую следуют из этого определения. Книга не беспокоится о том, существует ли производная, а она может отсутствовать даже у непрерывной кривой, например в остром углу модуля; к таким углам она возвращается в шестой главе.

Три правила переносятся на многомерный случай и будут использоваться постоянно. Правило суммы говорит, что наклон суммы — это сумма наклонов. Правило произведения говорит, что когда две величины перемножаются, изменение произведения складывается из изменения каждой, пока другая стоит на месте. Правило цепочки говорит, что когда одна функция подаётся в другую, наклоны вдоль цепочки перемножаются. Если обучение меняет настройку модели, что меняет её ответ, что меняет её ошибку, то общее влияние настройки на ошибку — это эти влияния, перемноженные вдоль цепочки, и на это правило будет опираться вся книга.

СУММА: НАКЛОНЫ СКЛАДЫВАЮТСЯ
ПРОИЗВЕДЕНИЕ: КАЖДЫЙ ПО ОЧЕРЕДИ
× 6



Наклоны перемножаются вдоль цепочки

Запомним

Einsum описывает почти любую операцию над тензорами короткой строкой букв, по одной на направление, где буквы, отсутствующие в результате, суммируются. Одна строка заменяет много библиотечных функций, одинаково работает в PyTorch и JAX и превращает переворот в перестановку букв. Производная — наклон прямой, касающейся кривой, лучшее прямолинейное подражание функции около точки, которое говорит, поднимается она рядом или падает. Правила суммы, произведения и цепочки распространяются на много измерений, и правило цепочки, перемножающее наклоны вдоль цепочки функций, книга будет использовать чаще всего.

Вопросы

1. В строке `einsum` буква встречается во входах, но не в выходе. Что происходит вдоль этого направления?
 - a. Оно копируется в результат без изменений
 - b. Оно складывается, суммируется прочь
 - c. Оно переворачивается, как при транспонировании
 - d. Возникает ошибка, потому что каждая буква должна быть в выходе
2. В точке, где наклон функции отрицателен, что делает функция рядом с этой точкой?

- a. Идёт вниз направо и вверх налево
 - b. Идёт вверх направо и вниз налево
 - c. Остаётся плоской в обе стороны
 - d. Обязательно имеет острый угол
3. Почему книга находит einsum таким полезным на практике?
- a. Потому что он быстрее любого другого умножения
 - b. Потому что он работает только в PyTorch
 - c. Потому что одна строка заменяет много библиотечных функций и одинаково работает в PyTorch и JAX
 - d. Потому что он автоматически вычисляет производные
4. Настройка модели влияет на её ответ, а ответ влияет на её ошибку. Как правило цепочки объединяет эти два влияния?
- a. Складывает их
 - b. Оставляет только большее
 - c. Перемножает их вдоль цепочки
 - d. Игнорирует первое

Ответы

1. **b)** По соглашению einsum любая буква, исчезающая на пути от входов к выходу, суммируется.
2. **a)** Положительный наклон означает подъём направо и спуск налево в маленькой окрестности; отрицательный наклон означает обратное.
3. **c)** Одна строка покрывает разные функции умножения с их правилами для форм и без изменений переносится между

библиотеками.

4. с) Когда одна функция подаётся в другую, их наклоны перемножаются, так что влияние настройки на ошибку — произведение влияний вдоль цепочки.

Задачи

1. Объясните своими словами, почему производную в точке называют лучшим прямолинейным подражанием функции там, используя линейку, касающуюся кривой.

Подсказка: Подумайте, что делают линейка и кривая совсем рядом с точкой касания.

Ответ: Линейка касается кривой в точке и наклонена туда же, куда направляется кривая, так что при крошечном шаге в любую сторону линейка и кривая почти совпадают. Никакая другая прямая не следует за кривой так близко около этой точки, а наклон этой прямой и есть производная.

2. Настройка меняет ответ модели втрое быстрее, чем движется сама настройка, а ответ меняет ошибку вдвое быстрее, чем движется ответ. По правилу цепочки, как быстро ошибка меняется с настройкой?

Подсказка: Правило цепочки перемножает влияния вдоль цепочки.

Ответ: В шесть раз быстрее: правило цепочки перемножает два влияния, три на два.

3. Опишите без символов, что говорит строка `einsum` для обобщённого скалярного произведения и почему после стрелки ничего нет.

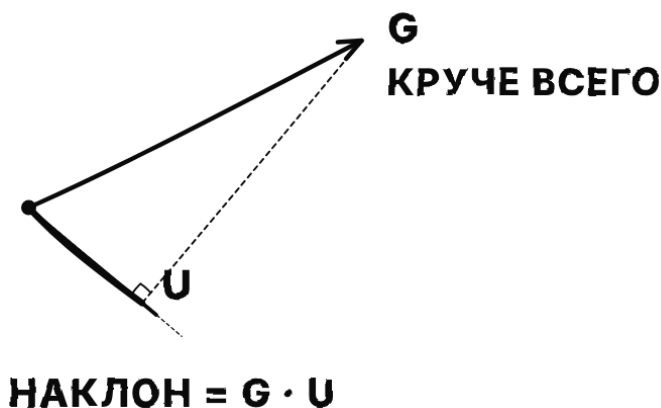
Подсказка: Какие буквы есть во входах, а какие в выходе?

Ответ: Входят две коробки с одними и теми же тремя буквами, так что они перемножаются поэлементно. Поскольку в выходе нет ни одной буквы, каждое направление суммируется прочь, и результат — одно число.

Наклоны во всех направлениях

Встаньте на травянистом склоне холма и задайте простой вопрос: земля здесь идёт вверх или вниз? На холме нет единственного ответа. Шагните на восток, и вы, возможно, поднимаетесь, шагните на север, и вы, возможно, спускаетесь, а шагните на северо-восток, и вы почти не меняете высоту. При одном числе на входе у наклона были только лево и право; при многих числах на входе направлений бесконечно много, и книга показывает, как справиться со всеми.

Простейший ход — только вдоль одной оси, когда все остальные координаты закреплены, как идти строго на восток. Наклон, который вы при этом чувствуете, называется частной производной. Сделайте это по разу для каждой оси, восток, север и так далее, для стольких направлений, сколько их у входа, и сложите все эти наклоны в один список. Этот список и есть градиент функции, и он так важен, что книга даёт ему собственный особый символ. Его можно представить как стрелку, лежащую на склоне.



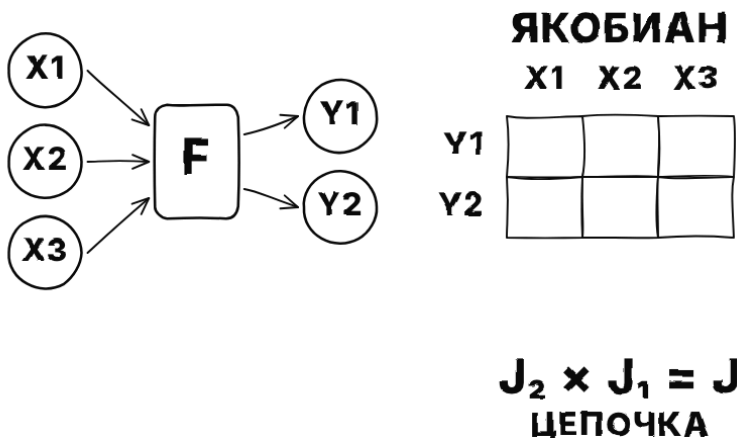
Наклон в любом направлении

А как насчёт ходьбы в любом другом направлении, скажем, немного на восток и много на север? Наклон вдоль произвольного направления называется производной по направлению. Поскольку любое движение можно разбить на части вдоль каждой оси, она оказывается просто скалярным произведением градиента и направления, в котором вы идёте. Так что одного градиента достаточно: по этой одной стрелке можно прочесть наклон в любом возможном направлении. Когда обучают приложение для фотографий, его миллионы настроек — оси огромного склона, и градиент говорит для каждой настройки сразу, в какую сторону его ошибки растут.

Опыт

Наклоните поднос на столе и катите по нему шарик с разных начальных направлений. Куда бы вы его ни толкнули, он в итоге скатывается вдоль одной определённой линии, самой крутой. Эта линия задана только наклоном подноса, а наклон любого другого направления — лишь доля от неё, и ровно это и значит, что градиент определяет все наклоны по направлениям.

Теперь пусть функция возвращает несколько чисел вместо одного. У каждого выхода свой градиент, и если сложить эти градиенты стопкой, по одному на строку, получится сетка, которую называют якобианом, с одной строкой на выход и одним столбцом на вход. При одном выходе якобиан — просто градиент, а при одном входе и одном выходе — обычная производная. Книга говорит, что это самый общий случай, который ей когда-либо понадобится.



Градиенты стопкой: якобиан

Якобианы сохраняют все хорошие привычки производных. Самое важное: когда две функции соединены в цепочку, якобиан цепочки — это произведение их якобианов, умножение матриц, и это правило цепочки, ставшее взрослым. Модель — длинная цепочка шагов, так что это правило позволяет собрать влияние каждой настройки на итоговую ошибку, перемножив сетки всех промежуточных шагов.

У градиентов и якобианов есть и геометрический смысл, как у касающейся линейки раньше. Около выбранной точки функция ведёт себя почти точно как плоская, прямолинейная версия самой себя: её значение в этой точке плюс градиент, приложенный к тому, насколько вы сдвинулись. Это лучшее

приближение первого порядка, результат, известный как теорема Тейлора. Книга проверяет его в коде на той же кривой в форме долины: на одну сотую шага от выбранной точки прямолинейная догадка и истинное значение совпадают с точностью до одной десятичной.



Прямолинейная догадка

Запомним

При многих входах наклон зависит от направления; движение вдоль одной оси даёт частную производную, а все они вместе дают градиент, стрелку. Наклон в любом направлении — скалярное произведение градиента и этого направления, так что градиент говорит всё о локальных наклонах. При нескольких выходах градиенты складываются в якобиан, по одной строке на выход, а функции в цепочке перемножают свои якобианы. Около любой точки функция ведёт себя как своё прямолинейное приближение, построенное из градиента, и это теорема Тейлора.

Вопросы

1. Почему знать градиент достаточно, чтобы знать наклон в любом направлении?
 - a. Потому что у функции многих входов только одно направление
 - b. Потому что любое движение разбивается на части вдоль осей, так что наклон — скалярное произведение градиента и направления
 - c. Потому что градиент всегда равен нулю
 - d. Потому что производная по направлению игнорирует направление

2. Функция берёт 5 чисел и возвращает 3 числа. Как выглядит её якобиан?
- a. Сетка из 3 строк и 5 столбцов, по одной строке на выход
 - b. Один список из 5 наклонов
 - c. Сетка из 5 строк и 3 столбцов, по одной строке на вход
 - d. Одно число
3. Две функции соединены в цепочку. Как получается якобиан цепочки?
- a. Сложением двух якобианов
 - b. Умножением двух якобианов, умножением матриц
 - c. Оставляют только якобиан последней функции
 - d. Их поэлементным умножением
4. Что говорит теорема Тейлора о функции рядом с выбранной точкой?
- a. Что она всюду равна своему значению в точке
 - b. Что она ведёт себя почти точно как прямолнейная версия, построенная из её значения и её градиента
 - c. Что она обязана быть многочленом
 - d. Что её градиент всюду одинаков

Ответы

1. **b)** Производная по направлению равна скалярному произведению градиента и направления движения, так что один градиент даёт каждый наклон по направлению.
2. **a)** У каждого выхода свой градиент, по одному на строку, так что у якобиана одна строка на выход и один столбец на вход.

3. б) Правило цепочки вырастает в умножение матриц якобианов соединённых функций.

4. б) Около точки функцию лучше всего подражает её значение плюс градиент, приложенный к смещению: приближение первого порядка.

Разобранные примеры

Прямолинейная проверка из книги Книга берёт кривую в форме долины $f(x) = x^2 - 1.5x$, чей наклон равен $2x - 1.5$, строит её прямолинейное приближение при $x = 0.5$ и сравнивает с настоящей кривой при $x = 0.51$. Какие значения она получает?

Значение в выбранной точке: $0,5$ в квадрате — это $0,25$, минус $1,5$ умножить на $0,5$, то есть $0,75$, даёт минус $0,5$.

Наклон в выбранной точке: 2 умножить на $0,5$ минус $1,5$ даёт минус $0,5$.

Прямолинейная догадка на одну сотую правее: минус $0,5$ плюс наклон, умноженный на $0,01$, то есть минус $0,5$ минус $0,005$, что равно минус $0,505$.

Истинное значение: $0,51$ в квадрате — это $0,2601$, минус $1,5$ умножить на $0,51$, то есть $0,765$, даёт минус $0,5049$.

Ответ: Догадка — минус $0,5050$, а истинное значение — минус $0,5049$: они расходятся лишь на одну десятичную.

Задачи

1. Объясните своими словами разницу между частной производной и производной по направлению, используя склон холма.

Подсказка: Одна о ходьбе вдоль одной оси, другая о любом выбранном направлении.

Ответ: Частная производная — наклон, который вы чувствуете, когда идёте строго вдоль одной оси, скажем, прямо на восток, держа все остальные координаты закреплёнными. Производная по направлению — наклон вдоль любого выбранного вами направления, например немного на восток и много на север; она сочетает частные производные в соответствии с тем, насколько вы сдвигаетесь вдоль каждой оси.

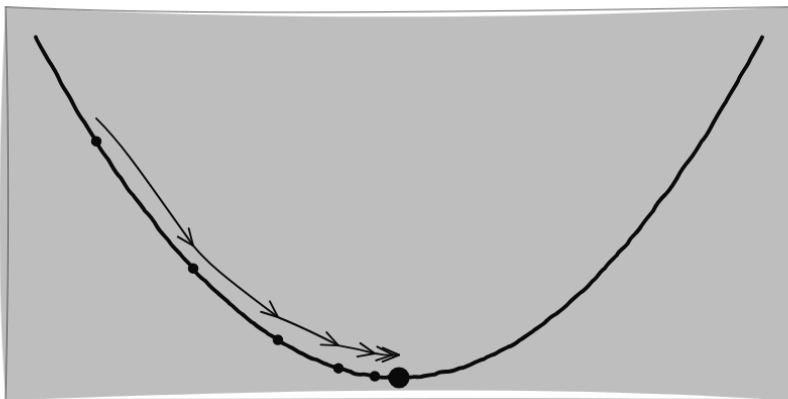
2. Почему правило цепочки для якобианов так важно для моделей, судя по этой части?

Подсказка: Думайте о модели как о длинной цепочке шагов.

Ответ: Модель — длинная цепочка шагов, и влияние каждой настройки на итоговую ошибку проходит через каждый промежуточный шаг. Поскольку якобиан цепочки — произведение якобианов её шагов, это влияние можно собрать, перемножив сетки всех шагов.

Спуск с горы в тумане

Представьте, что вы на горе в густом тумане и хотите добраться до дна долины. Долины вы не видите, только землю прямо под ногами. Поэтому вы нащупываете, в какую сторону земля падает круче всего, делаете маленький шаг туда и нащупываете снова. Шаг за шагом, никогда не видя всей местности, вы пробираетесь вниз. Это и есть градиентный спуск, метод в центре этой части, но сначала книга добавляет осторожное замечание о размерах якобианов.



МИНИМУМ

Нащупывая путь вниз

Возьмём простейший шаг модели: сетка весов, умноженная на входной вектор. Если смотреть на него как на функцию входа, его якобиан — просто сама сетка весов, и это успокаивает: наклон линейной машины — сама машина. Если смотреть на него как на функцию весов, вход функции сам является сеткой, и наклон становится коробкой с тремя направлениями: одно для каждого выхода и два для положения каждого веса. Приём книги — представить сетку весов развёрнутой в одну длинную строку чисел; тогда наклон снова становится обычной сеткой. Вот почему книга может записывать всё через векторы, не теряя общности.

РАЗВЁРНУТО

A	B	C	D	E	F
---	---	---	---	---	---

НАКЛОН

X	Y	Z	0	0	0
0	0	0	X	Y	Z

ХРАНИТСЯ ДВАЖДЫ

Небольшой блок разбирает это элемент за элементом. Каждый выход — сумма весов, умноженных на элементы входа, так что наклон выхода по одному из его весов — просто соответствующий элемент входа. Записать всю коробку наклонов в память значило бы поэтому хранить одни и те же числа входа снова и снова. Книга обещает, что в шестой главе это расточительство исчезнет, потому что на практике нам никогда не нужна сама коробка, а только то, что она делает, когда её применяют к другой коробке. Она также предупреждает, что аккуратные сетки на бумаге могут скрывать внутри огромные размеры, прежде всего размер пакета, так что реальная стоимость памяти может быть куда больше, чем подсказывает запись. Обучающий пакет из тысяч фотографий делает такие скрытые коробки громадными, вот почему настоящие программы обучения их никогда не строят.

Теперь главное событие: поиск самой низкой точки функции, её минимума. Книга предполагает, что функция даёт одно число и что вход может блуждать где угодно. В остальной книге входом будут параметры модели, а функция будет измерять, насколько плохо модель справляется с данными: постановка обучения с учителем из следующей главы. Минимизация не менее общая задача, чем максимизация: переверните кривую вверх ногами, и её высочайшие пики станут самыми глубокими долинами.

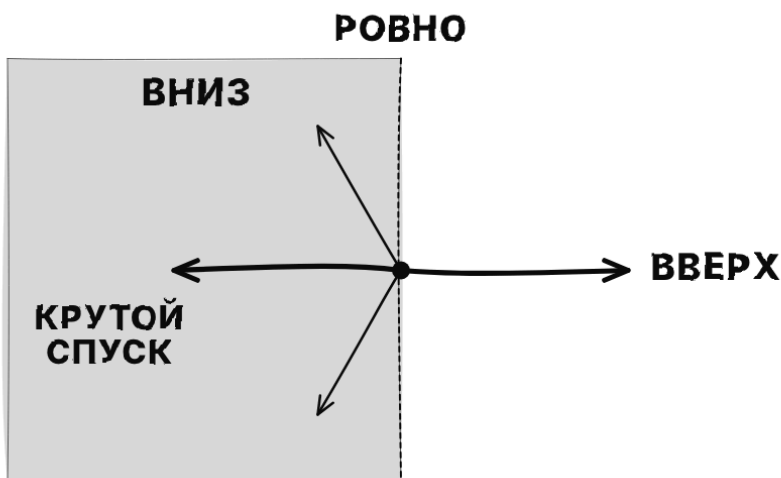


Шаг за шагом ко дну

Лишь в редких случаях самую низкую точку можно записать напрямую формулой; книга покажет один такой случай позже, для подгонки прямых. Обычно приходится искать шаг за шагом. Мы начинаем со случайной догадки и раз за разом делаем шаг, у которого две составляющие: его длина, называемая размером шага, и то, куда он указывает, его направление. В машинном обучении размер шага называют скоростью обучения, по причинам, которые прояснит следующая глава. Направление, которое при достаточно малом шаге может понизить функцию, называют направлением спуска, и если каждый шаг использует такое направление, а размеры шагов выбраны осторожно, поиск оседает в минимуме.

Опыт

Положите шарик на наклонную доску и слегка подталкивайте его в разных направлениях. Толкните его прямо вверх по склону, и он поднимается; толкните вбок вдоль склона, и он остаётся на той же высоте; толкните в любом направлении, которое хоть немного смотрит вниз, и он спускается. Толкните его прямо вниз по склону, и он спускается быстрее всего.



Какие направления ведут вниз

Ровно это говорит нам градиент, через наклон по направлению из прошлой части. Наклон в направлении — скалярное

произведение градиента и этого направления, то есть длина градиента, умноженная на то, насколько хорошо направление с ним выровнено. Длина градиента не зависит от выбранного направления, а каждое направление можно взять длины единица, так что решает только выравнивание. Любое направление, отклонённое от градиента больше чем на прямой угол, идёт вниз, а то, что направлено ровно против градиента, идёт вниз круче всех: направление наискорейшего спуска.

Соедините две идеи, и у вас алгоритм минимизации любой дифференцируемой функции, наискорейший градиентный спуск. Из текущей догадки шагните немного против градиента, затем вычислите градиент снова в новом месте и повторите. Выбором размера шага книга не занимается; она просто предполагает, что шаги достаточно малы, чтобы каждый понижал функцию. Обучение модели за чат-ассистентом — в своей основе та же прогулка, повторённая огромное число раз на местности с миллиардами направлений. Градиентный спуск быстр лишь настолько, насколько быстр наш способ вычислять градиенты, и общий метод для этого — предмет шестой главы.

Запомним

Со стороны входа якобиан сетки весов, умноженной на вектор, — сама сетка весов; со стороны весов это большая коробка, которую можно вытянуть в обычную сетку. Такие коробки повторяют много значений и скрывают размеры пакетов, поэтому практические методы их только применяют, но никогда не хранят. Чтобы минимизировать функцию, мы обычно ищем шаг за шагом, и у каждого шага есть длина, скорость обучения, и направление. Любое направление, отклонённое от градиента больше чем на прямой угол, идёт вниз, а ровно противоположное градиенту идёт вниз быстрее всего. Градиентный спуск повторяет маленькие шаги против градиента, и его скорость зависит от того, как быстро можно вычислять градиенты.

Вопросы

1. Среди всех направлений длины единица какое заставляет функцию падать быстрее всего?
 - a. Направление градиента
 - b. Любое направление под прямым углом к градиенту
 - c. Направление ровно против градиента
 - d. Все направления падают одинаково быстро
2. Почему книга говорит, что можно изучать минимизацию, ничего не теряя из максимизации?

- a. Потому что у функций никогда нет точек максимума
 - b. Потому что переворот функции вверх ногами превращает её пики в долины
 - c. Потому что градиентный спуск может находить только точки максимума
 - d. Потому что для максимизации не нужен градиент
3. Почему практические методы избегают хранить полную коробку наклонов для сетки весов, умноженной на вход?
- a. Потому что она повторяла бы одни и те же значения входа много раз и скрывала большие размеры пакета
 - b. Потому что она всегда равна нулю
 - c. Потому что её вообще нельзя вычислить
 - d. Потому что в ней меньше элементов, чем во входе
4. Что, по словам книги, решает, как быстро градиентный спуск может работать на практике?
- a. Цвет данных
 - b. Как быстро можно вычислять градиенты
 - c. Записана ли функция буквами
 - d. Сколько раз кривую перевернули вверх ногами

Ответы

1. **c)** Наклон решает только выравнивание с градиентом, и направление ровно против него даёт самый отрицательный наклон: направление наискорейшего спуска.
2. **b)** Минимизировать функцию — то же самое, что максимизировать её перевернутую версию, так что один метод покрывает обе задачи.

3. а) Каждый наклон по весу — просто соответствующий элемент входа, так что коробка полна повторов, а размеры пакета могут сделать её громадной; нужно лишь её действие на другую коробку.

4. б) Градиентный спуск эффективен лишь настолько, насколько эффективна процедура вычисления градиентов, предмет шестой главы.

Задачи

1. Путник на склоне пробует направление, которое образует с градиентом угол меньше прямого. Идёт путник вниз или вверх и почему?

Подсказка: Вспомните, что выравнивание с градиентом значит для наклона.

Ответ: Вверх. Направление, отклонённое от градиента меньше чем на прямой угол, положительно с ним выровнено, так что наклон в этом направлении положителен; вниз идут только направления, отклонённые больше чем на прямой угол.

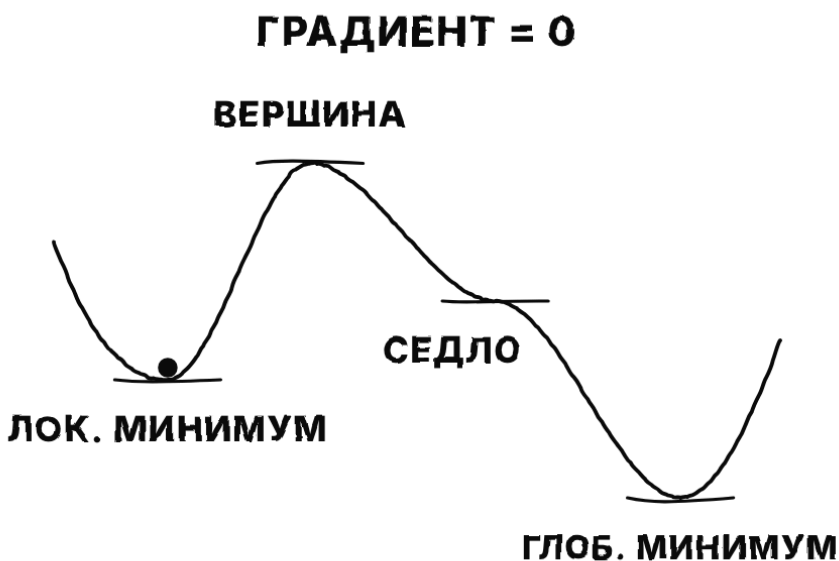
2. Опишите градиентный спуск своими словами, назвав две составляющие каждого шага и то, что происходит между шагами.

Подсказка: Вспомните путника в тумане.

Ответ: Начните с догадки. У каждого шага есть длина, размер шага или скорость обучения, и направление, которое при наискорейшем спуске ровно противоположно градиенту. После шага градиент вычисляется снова в новом месте, и процесс повторяется, причём шаги достаточно малы, чтобы каждый понижал функцию.

Где может закончиться прогулка

Налейте немного воды на смятый лист алюминиевой фольги и посмотрите, где она соберётся. Она не вся стекает в единственную самую низкую точку листа; она собирается в той маленькой вмятине, которая оказалась ближе всего. Градиентный спуск ведёт себя так же, и книга использует это, чтобы задать точный вопрос: когда прогулка вниз останавливается, где именно она остановилась? Автор добавляет, что если вы просто доверяете градиентному спуску, можете пропустить дальше; эта часть для тех, кто хочет знать, что он гарантирует.



Плоские места на смятой кривой

Локальный минимум — точка, которая ниже всего в маленькой окрестности вокруг неё, дно одной вмятины в фольге. В такой точке касательная горизонтальна, а вокруг неё земля поднимается. Горизонтальность схватывает более широкая идея, стационарная точка: любая точка, где градиент равен нулю, где наклона нет вовсе.

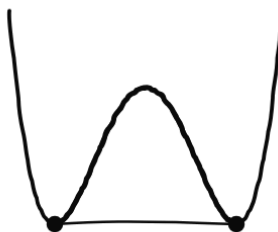
Не каждая плоская точка — вмятина. Вершина холма тоже плоская, как и седловая точка, место, где земля изгибается вверх в одном направлении и вниз в другом, как середина лошадиного седла или горный перевал. Рисунок в книге показывает такое седло и приглашает представить там горизонтальную касательную. Без дополнительных знаний о функции единственное, что можно доказать: градиентный спуск заканчивается в какой-то стационарной точке, а какой именно, зависит от того, где он начался.

Некоторые ландшафты дружелюбнее. Подумайте о простой чаше, как долина, вырезанная одной гладкой кривой: в ней нет сёдел и только одно дно, и это дно — самая низкая точка всего ландшафта, глобальный минимум. Из любой начальной точки каждый склон чаши ведёт к этому дну, так что градиентный спуск не может его пропустить. Книга обобщает чашу идей выпуклой функции.

ВЫПУКЛАЯ



НЕ ВЫПУКЛАЯ



Проверка ниткой

Опыт

Нарисуйте любую кривую в форме долины, выберите на ней две точки и натяните между ними нитку по прямой. Если кривая между двумя точками всегда остаётся ниже нитки, какие бы две точки вы ни выбрали, кривая выпуклая.

Теперь нарисуйте кривую с двумя впадинами и бугром между ними и попробуйте снова: при двух точках по обе стороны бугра кривая поднимается над ниткой, и она не выпуклая.

Выпуклость говорит вам, насколько лёгок поиск. На невыпуклом ландшафте градиентный спуск заканчивается в какой-то стационарной точке, и больше ничего нельзя сказать, не глядя на то, как меняются сами наклоны. На выпуклом ландшафте он достигает глобального минимума из любой начальной точки. А если кривая остаётся строго ниже каждой нитки, глобальный минимум ещё и единственный. Для невыпуклых задач плохая новость сурова: чтобы быть уверенным в глобальном минимуме, пришлось бы перезапускать поиск из каждой возможной начальной точки, что делает задачу в общем случае безнадежно трудной.

У этого глубокий исторический вес. Как покажет пятая глава, любая модель, которой стоит пользоваться, невыпуклая, так что её ландшафт обучения может быть полон вмятин и сёдел. Другие методы обучения, такие как метод опорных векторов, были привлекательны именно потому, что сохраняют выпуклый, безопасный поиск и при этом справляются с непрямолинейными закономерностями. И всё же сложные дифференцируемые модели замечательно работают на практике: начатое из разумной точки, обучение, как правило, заканчивается в местах, которые хорошо работают на реальных данных. Чат-ассистент, которым вы пользуетесь, был обучен на таком смятом ландшафте, и он работает, хотя никто не может обещать, что его обучение нашло самую низкую точку.

Запомним

Локальный минимум ниже своих ближайших окрестностей, и это стационарная точка, место с нулевым градиентом, как вершины холмов и седловые точки. В общем случае градиентному спуску гарантирована лишь остановка в какой-то стационарной точке, в зависимости от того, где он начинается. Выпуклая функция остаётся ниже прямой между любыми двумя своими точками, как чаша, и там градиентный спуск достигает глобального минимума откуда угодно, единственного, если выпуклость строгая. Полезные модели невыпуклы, в отличие от метода опорных векторов, но на практике их обучение приходит в точки, которые хорошо работают.

Вопросы

1. Градиентный спуск останавливается там, где градиент равен нулю. Почему это не гарантирует, что он нашёл минимум?
 - a. Потому что градиент никогда не равен ровно нулю
 - b. Потому что нулевой градиент бывает и на вершинах холмов, и в седловых точках
 - c. Потому что у минимумов всегда большой градиент
 - d. Потому что градиентный спуск работает только на чашах

2. Что можно сказать о градиентном спуске на выпуклой функции?
- a. Он достигает глобального минимума из любой начальной точки
 - b. Он всегда застревает в седловой точке
 - c. Его нужно перезапускать из каждой возможной точки
 - d. Он работает, только если начать в минимуме
3. Почему метод опорных векторов был привлекателен по сравнению с нейронными сетями?
- a. Ему не нужны данные
 - b. Он сохраняет выпуклый, безопасный поиск и при этом справляется с непрямолинейными закономерностями
 - c. Он всегда находит седловые точки
 - d. Его обучают более крупными шагами

Ответы

1. **b)** Любая точка с нулевым градиентом — стационарная, а к стационарным точкам относятся максимумы и седловые точки, а не только минимумы.
2. **a)** Как чаша, выпуклая функция ведёт каждый склон к своей самой низкой точке, так что градиентный спуск достигает глобального минимума откуда угодно.
3. **b)** Полезные нейронные модели невыпуклы, тогда как метод опорных векторов справляется с нелинейными закономерностями, сохраняя выпуклость оптимизации.

Задачи

1. Объясните своими словами разницу между локальным минимумом и глобальным минимумом, используя смятую фольгу.

Подсказка: Один о маленькой окрестности, другой о всём листе.

Ответ: Локальный минимум — дно одной вмятины: ниже всего рядом с ним, но не обязательно самая низкая точка листа.

Глобальный минимум — самая низкая точка всего листа. Вода, налитая на фольгу, обычно собирается в ближайшей вмятине, локальном минимуме, не обязательно глобальном.

2. Почему найти глобальный минимум невыпуклой функции градиентным спуском в общем случае так трудно?

Подсказка: Подумайте о том, как начальная точка решает, где закончится прогулка.

Ответ: На невыпуклом ландшафте то, где закончится прогулка, зависит от того, где она начинается, и она может остановиться в любой вмятине или на любом плоском месте. Чтобы быть уверенным в самой низкой точке, пришлось бы начинать из каждой возможной точки, что в общем случае безнадежно дорого.

3. Почему, несмотря на невыпуклость, нейронные сети всё же хорошо работают, по словам книги?

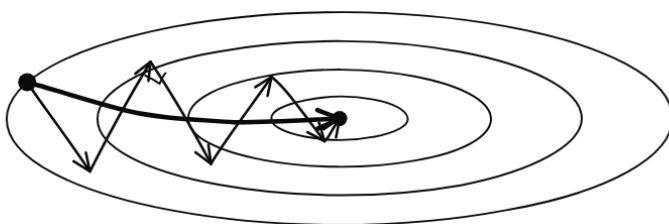
Подсказка: Подумайте о том, что происходит, когда обучение начинается из разумной точки.

Ответ: На практике, когда обучение начинается с разумной инициализации, градиентный спуск, как правило, заканчивается в точках, которые хорошо работают на реальных данных, даже если никто не может гарантировать, что это глобальный минимум.

Шарик с инерцией

Прокатите шарик по смятой простыне. Шарик, который слушался бы только наклона под собой в каждое мгновение, дёргался бы из стороны в сторону на каждой складке. Настоящий шарик так не делает: он сохраняет часть своей скорости и направления с прошлого мгновения и плавно перекачивается через мелкие бугорки. Эта память о прошлом движении, импульс, или момент, — первый приём, который книга показывает для ускорения градиентного спуска.

БЕЗ ПАМЯТИ



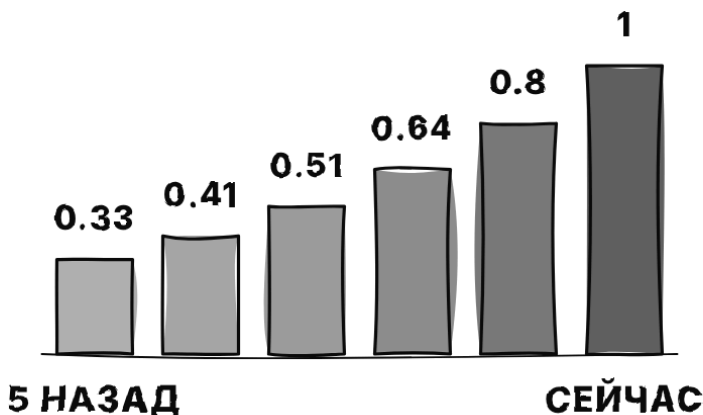
МОМЕНТ

Зигзаг против момента

Направление против градиента — самый крутой путь вниз, но лишь в крошечной окрестности текущей точки. Книга предупреждает, что, как покажет пятая глава, эти направления могут быть очень шумными, особенно в больших моделях. Придумано много методов, чтобы выбирать направления лучше и достигать дна быстрее. Книгу больше всего интересуют те, которым не нужны ни наклоны наклонов, которые дорого стоят, ни много дополнительных вычислений функции.

Представьте простой градиентный спуск как шарик, катящийся вниз без памяти: его путь может быть беспорядочным, потому что каждый новый градиент может указывать куда-то совсем в другую сторону. Книга даже замечает, что при идеальном размере шага на функции в форме чаши два соседних градиента стоят под прямым углом друг к другу, так что путь идёт зигзагом. Момент сглаживает зигзаг, сохраняя долю предыдущего шага и прибавляя её к новому. Одна настройка решает, какую часть прошлого сохранять. Если развернуть правило, видно, что старые шаги всё ещё учитываются, но каждый чуть меньше следующего за ним, затухая тем сильнее, чем дальше назад он лежит.

ХРАНИТЬ 0.8 ПРОШЛОГО



Старые шаги весят меньше

Рисунок в книге сравнивает оба на волнистой кривой, простой градиентный спуск и версию с моментом, запущенные из одной точки. Известно, что момент ускоряет обучение, сглаживая путь. Другой распространённый приём даёт каждому параметру собственный размер шага, подстроенный под то, насколько велики были его градиенты. Широко используемый метод под названием Adam сочетает несколько таких идей.

Опыт

Наполните чашу водой и мешайте её крошечными движениями, всё время меняя направление: вода почти не двигается. Теперь мешайте ровно в одну сторону несколько секунд: вода начинает кружиться и продолжает кружиться даже после того, как вы остановились. Маленькие толчки, согласные друг с другом, складываются, а толчки, которые всё время гасят друг друга, нет: вот что момент делает с шагами градиентного спуска.

Большое практическое достоинство Adam в том, что он хорошо работает без долгой возни с настройками. Настройки, которые выбирает пользователь, а не выучивает градиентный спуск, называют гиперпараметрами, и для Adam значения по умолчанию из библиотеки обычно хорошее начало. Многие модели, которые вы встречаете каждый день, от приложений для фотографий до чат-ассистентов, обучены Adam или его близким родственником. Создать новый оптимизатор, который сможет заменить Adam или его вариант AdamW в роли выбора по умолчанию, остаётся открытой исследовательской задачей. Цена этих ускорений — память: момент, например, должен помнить предыдущий шаг, что удваивает память, нужную самому оптимизатору, хотя вычисление градиента обычно требует памяти куда больше.

Глава заканчивается разделом под названием «От теории к практике». Вместо классических упражнений книга предлагает самостоятельный путь через две библиотеки программирования, JAX и PyTorch, с решениями на сайте книги, и предупреждает,

что веб-ссылки могли переехать к тому времени, когда вы их читаете. Начальный блок — NumPy, стандартная библиотека для работы с коробками чисел: создавать их, вырезать из них ломтики и преобразовывать. Автор советует освоиться с ней, прежде всего с вырезанием ломтиков, и указывает на открытую коллекцию из ста несложных упражнений, чтобы проверить себя.

Запомним

Самое крутое направление верно лишь очень локально и может быть шумным, поэтому более быстрые методы выбирают направления лучше, не прибегая к дорогим дополнительным производным. Момент сохраняет затухающую долю прошлых шагов, сглаживая зигзагообразный путь простого градиентного спуска и ускоряя обучение. Adam сочетает момент с размером шага для каждого параметра и хорошо работает с гиперпараметрами по умолчанию, поэтому остаётся стандартным выбором. Эти приёмы стоят памяти, а практический путь начинается с хорошего знания NumPy.

Вопросы

1. Что момент добавляет к простому градиентному спуску?
 - а. Долю предыдущего шага, прибавленную к новому, так что путь сглаживается

- b. Вторую производную, вычисляемую на каждом шаге
 - c. Случайный прыжок в новую начальную точку
 - d. Большой размер шага для каждого параметра
2. При моменте насколько ещё учитывается шаг, сделанный давно?
- a. Ровно столько же, сколько самый новый шаг
 - b. Совсем не учитывается уже через один шаг
 - c. Чуть меньше с каждым шагом назад, затухая
 - d. Больше, чем самый новый шаг
3. Почему Adam так популярен как оптимизатор по умолчанию?
- a. Ему вообще не нужны градиенты
 - b. Он хорошо работает без долгой возни, и настройки по умолчанию из библиотеки — хорошее начало
 - c. Он не использует памяти, кроме параметров
 - d. Он всегда находит глобальный минимум
4. В чём главная цена использования момента?
- a. Он делает путь более беспорядочным
 - b. Он должен помнить предыдущий шаг, удваивая память оптимизатора
 - c. Он требует, чтобы функция была выпуклой
 - d. Он перестаёт работать на больших моделях

Ответы

1. a) Момент сохраняет долю предыдущего шага и прибавляет её к новому, сглаживая зигзаг простого градиентного спуска.

2. c) Если развернуть правило, видно, что старые шаги всё ещё учитываются, каждый чуть меньше следующего за ним, затухая тем сильнее, чем дальше назад он лежит.

3. b) Adam относительно устойчив к выбору своих гиперпараметров, и значения по умолчанию в большинстве библиотек — хорошая отправная точка.

4. b) Момент хранит предыдущий шаг, что удваивает память оптимизатора, хотя вычисление градиента обычно требует гораздо больше памяти.

Задачи

1. На примере чаши с водой, которую мешают, объясните, почему момент помогает, когда последовательные градиенты всё время меняют направление.

Подсказка: Подумайте о том, что происходит с толчками, которые согласны друг с другом, и с толчками, которые гасят друг друга.

Ответ: Толчки в меняющихся направлениях в основном гасят друг друга, так что с моментом зигзагообразные части шагов сжимаются, а части, согласные от шага к шагу, складываются. Путь становится более гладким и увереннее продвигается вниз.

2. Что такое гиперпараметр и чем он отличается от параметров, которые выучивает градиентный спуск? Приведите в пример настройку момента.

Подсказка: Кто его выбирает: пользователь или обучение?

Ответ: Гиперпараметр — настройка, которую пользователь выбирает до обучения, а параметры подстраивает градиентный спуск. Доля прошлых шагов, которую сохраняет момент, — гиперпараметр: его выбирает пользователь, и он не выучивается.

От учебного кода к настоящим данным

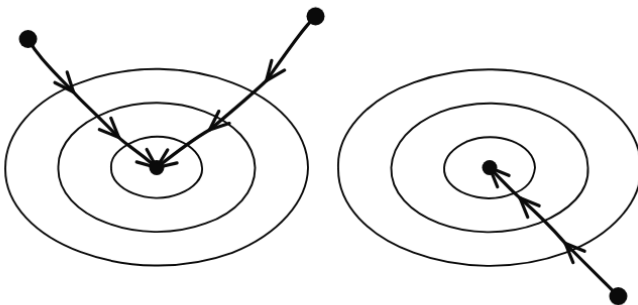
Представьте, что вы пытаетесь написать вручную список правил, который отличит машину от велосипеда на любой фотографии: цвет, контур, число колёс, тени, угол съёмки. Вы никогда не закончите. А вот собрать десять тысяч фотографий с подписями «машина» или «велосипед» легко. Этот контраст, правила невозможны, а примеры легки, — отправная точка третьей главы, но сначала книга заканчивает свои практические советы.

У NumPy, при всём его влиянии, два слабых места для наших целей: сам по себе он мало использует графические чипы и не умеет вычислять градиенты автоматически. JAX копирует манеру работы NumPy и добавляет поддержку такого оборудования, автоматические градиенты и дополнительные преобразования, например такое, которое превращает функцию, написанную для одного входа, в функцию, обрабатывающую целый пакет. У PyTorch тоже ядро в духе NumPy, с некоторыми изменениями имён и поведения, плюс высокоуровневые инструменты для построения моделей. Автор предлагает пролистать базовые страницы о массивах в обеих библиотеках, чтобы увидеть, как много у них общего с NumPy, а высокоуровневые части для построения моделей пока пропустить.

Затем идёт практическое упражнение, которое нужно сделать трижды, по разу в каждой библиотеке, и оно должно занять лишь минуты, когда вы знаете синтаксис. В нём используется функция

двух чисел с несколькими холмами и долинами, и автор добавляет откровенное замечание: эту единственную функцию сгенерировал ChatGPT, и больше ничего в книге не написано языковой моделью, оговорка, которая, по мнению автора, становится важной. Сначала вы пишете функцию так, чтобы она принимала целый пакет точек сразу, потом пишете её градиент вручную, поскольку автоматические градиенты появятся лишь позже. Далее вы запускаете градиентный спуск из нескольких начальных точек и рисуете пути, а в конце добавляете момент и смотрите, как длина градиента сжимается к нулю по мере того, как поиск достигает плоского места.

СТАРТ РЕШАЕТ ДОЛИНУ



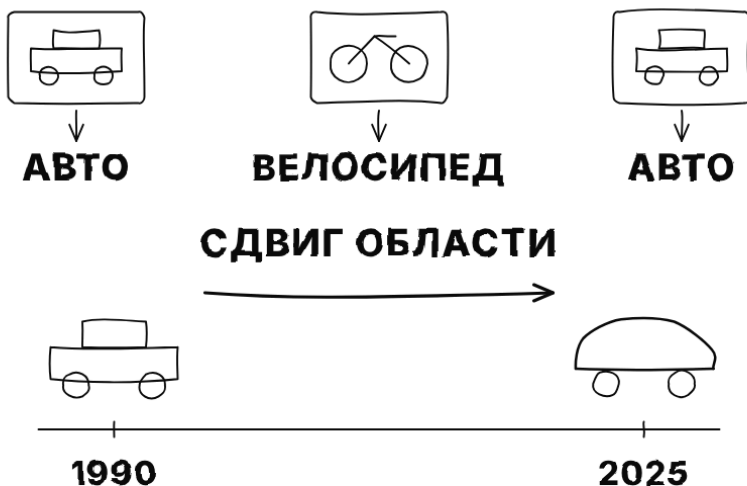
Разные старты, разные долины

Опыт

Если компьютер под рукой, всё же сначала попробуйте суть упражнения на бумаге. Нарисуйте ландшафт из нескольких холмов и долин горизонталями, как на туристической карте, и отметьте три начальные точки. От каждой проведите короткие стрелки, всегда направленные прямо вниз поперёк горизонталей: вы увидите, что разные начальные точки заканчивают в разных долинах, как покажут и графики упражнения.

Теперь третья глава делает обучение с учителем точным: наборы данных, функции потерь, идея минимизировать среднюю ошибку на данных, допущения, стоящие за всем этим, и версия, построенная на вероятности. Автор называет эту короткую главу позвоночником остальной книги. Она начинается ровно там, где нас оставила задача с машиной и велосипедом: задачи, которые нельзя запрограммировать вручную, но для которых легко собрать примеры, от распознавания речи до управления роботом. Набор данных — это тогда просто коллекция пар, каждая — вход вместе с выходом, который мы для него хотим.

НАБОР ДАННЫХ: ВХОД → ОТВЕТ



Пары из меняющегося мира

Одно допущение о том, как собраны пары, значит чрезвычайно много, и книга объясняет его на машинах. Примеры должны извлекаться одинаковым образом из устойчивого, неменяющегося источника. Если вы собираете фотографии машин несколько месяцев, это выполняется; если собираете их несколько десятилетий, дизайн машин меняется, и обученная на них модель будет отказывать на новых моделях или работать плохо. Их также нужно собирать без перекаса, чтобы коллекция честно представляла всё, с чем модель встретится. Фотографии, снятые только рядом с автосалоном одной марки, были бы полны этой марки и бедны всеми остальными. И ответ зависит от контекста: фотографии машин из Италии могут подойти модели,

которую используют в Риме или Милане, но не той, что используют в Токио.

Книга настаивает, что это не проверяется один раз и забывается; за этим нужно следить всю жизнь модели. Если машины на дорогах медленно меняются и никто этого не замечает, качество модели тихо распадается, проблема, которую называют сдвигом области. Рекомендательная система, как та, что выбирает, что предложат вам приложения дальше, даже меняет само поведение, на котором учится, потому что люди реагируют на её предложения, создавая петли обратной связи, которые требуют постоянной перепроверки. Для сегодняшних огромных языковых моделей, обученных на таких обширных и разнообразных данных, становится трудно даже сказать, какие задачи считать знакомыми, а какие новыми.

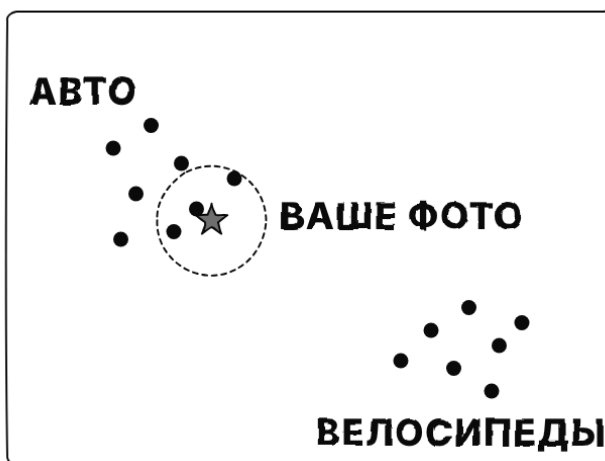
Рисунок главы добавляет картину, которую стоит запомнить.

Обученная модель преобразует свой вход шаг за шагом, и внутренние версии данных, которые она создаёт, часто имеют осмысленную геометрию: изображения с похожим смыслом попадают в близкие точки внутреннего пространства.

Превращение сырых данных, например фотографий, в точки, где расстояние означает сходство, называют эмбедингом данных.

Именно поэтому ваше приложение для фотографий может найти снимки, похожие на выбранный: похожие фотографии лежат рядом в его внутреннем пространстве.

ЭМБЕДДИНГ



Похожее лежит рядом

Запомним

JAX и PyTorch сохраняют манеру работы NumPy, но добавляют поддержку графических чипов и автоматические градиенты, а упражнение книги реализует градиентный спуск на холмистой функции во всех трёх библиотеках. Обучение с учителем — для задач, которые мы не можем запрограммировать вручную, но можем показать на примерах, а набор данных — коллекция пар вход-выход. Примеры должны идти из устойчивого источника и собираться без перекоса, что зависит от контекста и должно проверяться всю жизнь модели, чтобы поймать сдвиг области и петли обратной связи. Обученные модели часто встраивают данные так, что похожие элементы лежат рядом во внутреннем пространстве.

Вопросы

1. Модель распознавания машин обучена на фотографиях, собранных за последние сорок лет. Какую проблему предсказывает книга?
 - a. Модель будет слишком мала, чтобы хранить столько фотографий
 - b. Дизайн машин менялся со временем, так что источник не был устойчивым, и модель может отказывать на новых моделях
 - c. Фотографии будут слишком похожи друг на друга

- d. Модель будет распознавать только велосипеды
2. Почему фотографии, снятые только рядом с автосалоном одной марки, были бы плохим набором данных?
- a. Потому что они были бы собраны с перекосом и не представляли бы всех машин, с которыми встретится модель
 - b. Потому что фотографии у автосалонов всегда размыты
 - c. Потому что фотографии охватывали бы слишком долгий период
 - d. Потому что набор данных обязан содержать и велосипеды
3. Что значит, что обученная модель встраивает свои данные?
- a. Она скрывает данные, чтобы никто не мог их прочесть
 - b. Она превращает сырые данные в точки, где близость означает сходство
 - c. Она хранит каждую фотографию дважды
 - d. Она убирает из данных все числа
4. Почему допущения о наборе данных нужно проверять всю жизнь модели, а не один раз?
- a. Потому что библиотеки каждый год меняют названия
 - b. Потому что мир, с которым встречается модель, может медленно меняться, а некоторые системы даже меняют поведение, на котором учатся
 - c. Потому что наборы данных со временем теряют примеры
 - d. Потому что градиенты растут после обучения

Ответы

1. **b)** Примеры должны идти из устойчивого, неменяющегося источника; за десятилетия дизайн машин меняется, так что модель может отказывать на новых моделях или работать плохо.
2. **a)** Примеры нужно собирать без перекоса; рядом с одним автосалоном эта марка была бы представлена избыточно, а другие марки и модели отсутствовали бы.
3. **b)** Внутри обученной модели элементы с похожим смыслом попадают в близкие точки, и это называют эмбедингом данных.
4. **b)** Медленные изменения вызывают сдвиг области, а системы вроде рекомендательных создают петли обратной связи, меняя поведение людей.

Задачи

1. Книга просит реализовать градиентный спуск на холмистой функции двух чисел в NumPy, JAX и PyTorch. Перечислите четыре шага упражнения и скажите, что вы ожидаете увидеть на последнем.

Подсказка: Вспомните функцию для пакетов, градиент вручную, пути и момент.

Ответ: Сначала написать функцию так, чтобы она обрабатывала целый пакет точек сразу; во-вторых, написать её градиент вручную; в-третьих, запустить градиентный спуск из нескольких начальных точек и нарисовать пути; в-четвёртых, добавить момент и построить график длины градиента. На последнем шаге длина градиента должна сжиматься к нулю по мере того, как поиск приближается к плоскому месту.

2. Набор данных с машинами, собранный в Италии, используется для модели, развёрнутой в Милане, и другой, развёрнутой в Токио. Объясните, почему допущения могут выполняться для одной и не выполняться для другой.

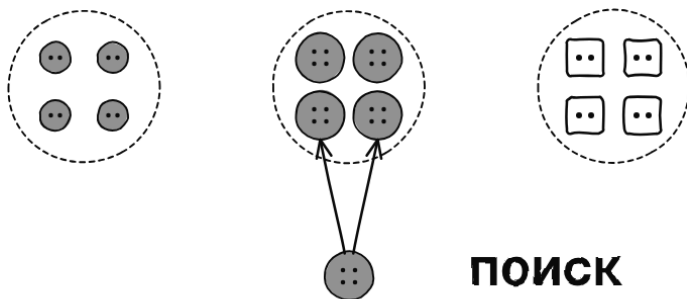
Подсказка: Спросите себя, похожи ли машины, с которыми встречается модель, на машины в наборе данных.

Ответ: Машины на дорогах Милана похожи на те, что сфотографированы в других местах Италии, так что набор данных представляет то, с чем встретится эта модель. В Токио набор марок и моделей другой, так что набор данных больше не представляет машины, которые видит модель, и её качество может пострадать.

Обучение без меток и повторное использование выученного

Высыпьте коробку разных пуговиц на стол и разложите их по кучкам, причём никто не говорит вам, какими должны быть кучки. Вы, скорее всего, сгруппируете их по цвету, размеру или числу дырочек, и кучки будут осмысленными, хотя никто не дал вам ни одной метки. В обучении с учителем у каждого примера всегда был ответ; эта часть смотрит на то, что происходит, когда ответов нет, и на то, как модель, обученную один раз, можно использовать много раз.

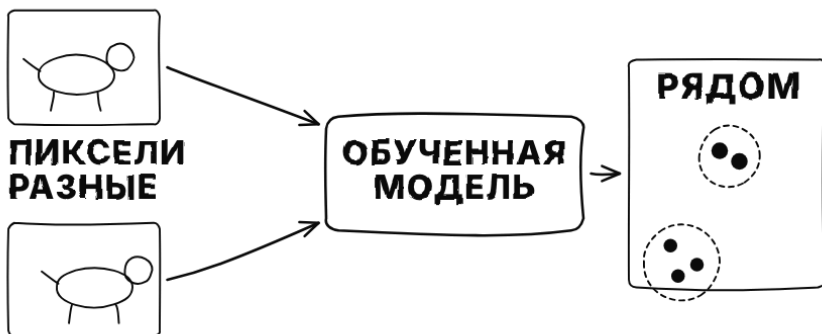
КЛАСТЕРИЗАЦИЯ



Кучки без меток

Когда в наборе данных есть входы, но нет целевых ответов, книга говорит об обучении без учителя. Классическая задача — кластеризация: сгруппировать данные так, чтобы элементы внутри кучки были похожи, а элементы из разных кучек нет, как с пуговицами. Другая — поиск: просмотреть огромную коллекцию в поисках немногих элементов, наиболее похожих на то, что попросил человек. Попросить приложение для фотографий найти ещё снимки, похожие на выбранный, — ровно такой поиск.

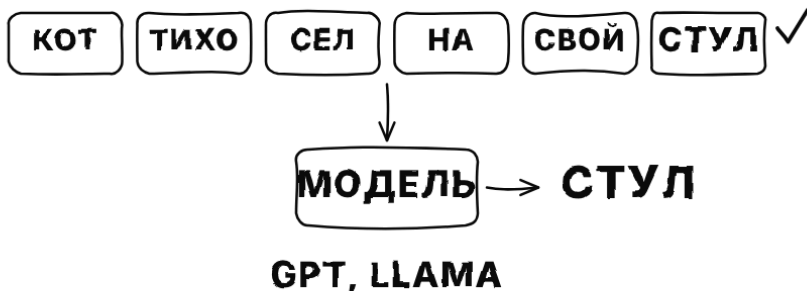
Однако с изображениями сходство на удивление трудная вещь. Сравнение фотографий пиксель за пикселем не работает, потому что крошечный сдвиг камеры меняет миллионы пикселей, хотя картинка значит то же самое. Выход, который предлагает книга, — предобученная модель, уже обученная на довольно общей задаче, например называть, что на фотографии. Её внутренние состояния можно читать как точки в пространстве с очень многими направлениями, и часто похожие вещи оказываются там рядом, как мы видели с эмбедингами. Так что можно взять эти внутренние точки и запустить на них обычный метод кластеризации, и кучки теперь следуют смыслу, а не пикселям.



Пиксели спорят, смысл согласен

А если предобученной модели нет? Тогда популярный вариант, который называют самообучением, или самоконтролируемым обучением, изобретает задачу с учителем из неразмеченных данных. Например, при большой коллекции текста всегда можно обучить программу угадывать, как продолжается короткий кусок текста, потому что настоящее продолжение тут же, в тексте. Открытие, что сети, обученные таким способом, выучивают ещё и отличные эмбединги текста, глубоко повлияло на область. Книга добавляет предупреждение: текст масштаба всего интернета полон предвзятостей и грубого содержания, модели впитывают эти предвзятости, и это одна из главных претензий к закрытым фундаментальным моделям.

ОТВЕТ БЫЛ В ТЕКСТЕ



Задача, спрятанная в тексте

Сегодняшние большие языковые модели, такие как GPT или Llama, — современная форма этой идеи: их обучение всегда начинается с угадывания следующего куска текста, и они входят в число фундаментальных моделей, встреченных раньше. Рисунок в книге показывает три способа использовать такую модель. Использование без примеров, zero-shot, просто напрямую задаёт модели новый вопрос. Промпт с несколькими примерами, few-shot, сначала помещает в промпт горстку разобранных примеров новой задачи, и оба способа работают, только если модель хорошо обобщает. Дообучение идёт дальше и подстраивает параметры модели градиентным спуском на маленьком наборе данных для новой задачи, во многом как обучение с нуля, только короче. Когда вы вставляете в чат-

ассистент два образца ответов и просите третий в том же стиле, вы делаете промпт с несколькими примерами.

Дообучение сегодня даётся легко благодаря большим открытым коллекциям общедоступных моделей. Оно может подстраивать все параметры, или лишь малую часть, или несколько добавленных, семейство приёмов, называемое экономным по параметрам дообучением, которое автор оставляет для следующего тома. Когда модель выучивает задачу только по примерам в своём промпте, без всякого дообучения, книга называет это обучением в контексте. Существует много других вариантов: наборы данных, размеченные лишь частично, которых касается двенадцатая глава; связанные наборы данных или одни и те же данные, сдвигающиеся со временем; и такие задачи, как адаптация к новым областям, обучение учиться, непрерывное обучение, обучение расстояниям и даже заставить модель забыть. Книга лишь намекает на них и оставляет несколько для второго тома.

Запомним

Без целевых ответов обучение без учителя группирует данные в кластеры или находит самые похожие элементы. Для изображений расстояния между пикселями не работают, но внутренние представления предобученной модели помещают похожие вещи рядом, так что кластеризация может работать на них. Самообучение изобретает собственную задачу, например угадывать продолжение текста, и так начинают своё обучение языковые модели вроде GPT и Llama, вместе с предвзятостями. Фундаментальную модель можно использовать без примеров, с несколькими примерами в промпте или дообучить градиентным спуском, полностью или экономными по параметрам методами.

Вопросы

1. Почему сравнение двух фотографий пиксель за пикселем — плохой способ судить об их сходстве?
 - a. Потому что у фотографий слишком мало пикселей
 - b. Потому что крошечный сдвиг камеры меняет миллионы пикселей, а смысл остаётся тем же
 - c. Потому что пиксели нельзя хранить как числа
 - d. Потому что у похожих фотографий всегда одинаковые пиксели

2. Как самообучение может учиться на куче текста вообще без меток?
 - a. Попросив людей разметить каждое предложение
 - b. Изобретя задачу, ответ на которую уже есть в тексте, например угадывать, как продолжается кусок текста
 - c. Случайно разбив текст на кучки
 - d. Дообучив уже существующую модель
3. Вы даёте чат-ассистенту два примера задачи в промпте, а потом просите третий, не меняя модель. Как это называется?
 - a. Дообучение
 - b. Кластеризация
 - c. Промпт с несколькими примерами
 - d. Экономное по параметрам дообучение
4. Что отличает дообучение от использования без примеров и с несколькими примерами?
 - a. Дообучение меняет параметры модели градиентным спуском на маленьком наборе данных
 - b. Дообучению вообще не нужны данные
 - c. Дообучение работает только на данных без меток
 - d. Дообучение — то же самое, что задать вопрос напрямую

Ответы

1. b) Малые возмущения меняют огромное число пикселей, так что расстояния между пикселями мало говорят о смысле; внутренние представления предобученной модели работают лучше.

2. b) Настоящее продолжение тут же, в тексте, так что его предсказание даёт задачу с учителем бесплатно.

3. c) Поместить горстку примеров в промпт — это промпт с несколькими примерами; параметры не меняются, и книга называет это обучением в контексте.

4. a) Использование без примеров и с несколькими примерами оставляет модель как есть, а дообучение подстраивает её параметры на примерах новой задачи.

Задачи

1. У вас тысячи фотографий из отпуска без меток, и вы хотите сгруппировать их по тому, что на них изображено. Опишите, как идеи книги позволяют это сделать.

Подсказка: Пикселей недостаточно; подумайте, где живёт осмысленное сходство.

Ответ: Пропустите каждую фотографию через предобученную модель, например обученную называть объекты, и возьмите её внутреннее представление, точку в многомерном пространстве, где похожие вещи оказываются рядом. Затем запустите на этих точках обычный метод кластеризации, чтобы группы следовали смыслу, а не пикселям.

2. Объясните своими словами предупреждение книги об обучении на тексте масштаба всего интернета.

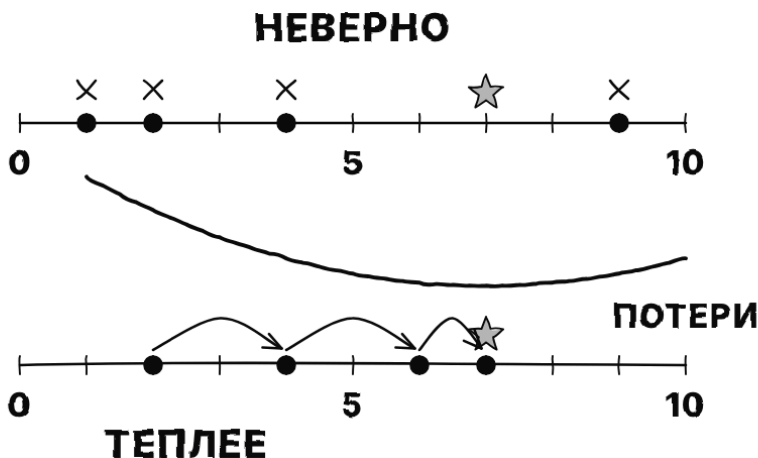
Подсказка: Подумайте о том, что содержит текст и чему модель из него учится.

Ответ: Текст из интернета содержит предвзятости и грубое содержание. Модель, обученная подражать этому тексту, впитывает эти предвзятости, и это одна из главных претензий к закрытым

фундаментальным моделям, чьи обучающие данные нельзя проверить.

Оценка догадки

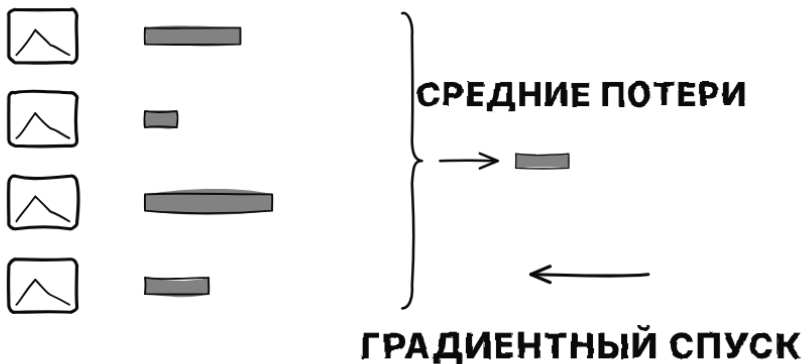
Сыграйте с другом в угадайку: он загадывает число, вы угадываете, а он отвечает только «верно» или «неверно». Догадка за догадкой вы слышите «неверно» и не имеете понятия, приближаетесь ли вы. Теперь поменяйте правило: он говорит «теплее» или «холоднее». Внезапно каждая догадка учит вас, в какую сторону двигаться, и вы быстро выходите на цель. В этой разнице вся суть этой части: модель может учиться, только если её ошибки оцениваются так, что видно, в какую сторону двигаться.



Неверно или теплее?

Оценку называют функцией потерь. Она сравнивает предсказание модели с целевым ответом и даёт одно число, где меньше значит лучше, так что меньшие потери всегда означают лучшее предсказание. И она должна быть дифференцируемой, чтобы мы могли сказать, как она меняется при сдвиге предсказания. Книга подчёркивает, что функция потерь упаковывает наше понимание задачи и наши предпочтения среди возможных ответов в одну шкалу, с которой может работать оптимизатор.

ПРИМЕРЫ ПОТЕРИ



Средние потери, сдвинутые вниз

Обучение тогда становится ясной математической задачей: найти модель, чьи потери, усреднённые по всем примерам набора данных, как можно меньше. По историческим причинам это

называют минимизацией эмпирического риска, где риск — просто другое слово для потерь, а эмпирический значит измеренный на данных, которые у нас действительно есть. На практике мы не перебираем все мыслимые функции; у модели фиксированная форма с подстраиваемыми параметрами, и градиентный спуск настраивает эти параметры, чтобы снизить средние потери. Когда обучают приложение для фотографий, происходит буквально это: его параметры снова и снова подкручивают, чтобы снизить его средний балл ошибки на миллионах размеченных фотографий.

Опыт

Спрячьте монету под одной из чашек в ряду и попросите кого-нибудь её найти, отвечая сначала только «да» или «нет» на каждую догадку, а потом говоря, на сколько чашек он промахнулся. С «да» или «нет» он должен проверять чашку за чашкой наугад. С «насколько далеко» он может идти прямо к цели. Первое — потери, которые никогда не говорят, куда идти; второе — потери с наклоном.

Книга показывает, почему дифференцируемость — не формальность. Возьмём задачу с двумя возможными ответами, скажем «да» или «нет», и модель, которая отвечает знаком числа, которое вычисляет. Самая естественная оценка — просто верно или неверно: ноль за верный ответ, единица за неверный. Она совпадает с нашим чувством правильности, но бесполезна для обучения: почти всюду её наклон равен нулю, так что градиентный спуск не получает подсказки и застревает там, где начал.



У ступеньки нет наклона

Менее интуитивная, но более гладкая величина — отступ, который сравнивает число модели с целью: положительный, когда знак согласен с верным ответом, отрицательный, когда нет, и тем более положительный, чем уверенней и правильной модель. Поскольку отступ меняется гладко, построенные на нём потери дают полезные наклоны. Одни такие потери, кусочно-линейные потери, или hinge loss, которыми обучали метод опорных векторов, равны нулю, как только отступ достаточно велик, и растут ровно по мере того, как отступ сжимается или становится неверным. Пример показывает постоянное напряжение: потери должны схватывать то, что мы понимаем под хорошей работой, и в то же время быть удобными для численной оптимизации.

Небольшой блок прослеживает эти идеи до российского учёного Владимира Вапника и области статистической теории обучения. Эта область изучает, насколько хорошо средние потери на ограниченном наборе данных заменяют среднее по всем возможным данным, для хорошо устроенных семейств моделей. Современные нейронные сети поставили её в тупик: они продолжают хорошо обобщать долго после того, как теория говорила, что они должны начать запоминать, и это открыло новые направления исследований.

Запомним

Функция потерь оценивает предсказание относительно его цели одним числом, меньшим для лучшего, и должна быть дифференцируемой, чтобы говорить, в какую сторону подстраивать. Обучение — это минимизация эмпирического риска: настройка параметров модели градиентным спуском так, чтобы средние потери на наборе данных стали как можно меньше. Простая оценка «верно или неверно» бесполезна для обучения, потому что её наклон почти всегда равен нулю. Гладкие величины, такие как отступ, и построенные на них потери вроде кусочно-линейных дают наклоны, по которым можно идти, и это показывает напряжение между тем, что мы понимаем под хорошим, и тем, что можно оптимизировать.

Вопросы

1. Почему простая оценка «верно или неверно» бесполезна для обучения градиентным спуском?
 - a. Потому что она даёт слишком большие числа
 - b. Потому что её наклон почти всюду равен нулю, так что градиентный спуск не получает подсказки и застревает
 - c. Потому что её нельзя усреднить по набору данных
 - d. Потому что она награждает неверные ответы
2. Чего требует минимизация эмпирического риска?
 - a. Модели с наименьшими средними потерями по примерам набора данных
 - b. Модели с наибольшим числом параметров
 - c. Модели, которая никогда не делает ни одной ошибки
 - d. Модели с наибольшим отступом на одном примере
3. Модель отвечает «да» или «нет» знаком числа, которое вычисляет. Когда её отступ отрицателен?
 - a. Когда модель права и очень уверена
 - b. Когда знак числа модели не согласен с правильным ответом
 - c. Когда число модели равно ноль
 - d. Никогда; отступы всегда положительны
4. Какую загадку современных нейронных сетей книга упоминает в связи со статистической теорией обучения?
 - a. Их нельзя обучать градиентным спуском
 - b. Они продолжают хорошо обобщать долго после того, как теория ожидала, что они начнут запоминать
 - c. Они работают только с оценкой «верно или неверно»

d. Они никогда не используют функции потерь

Ответы

1. **b)** Оценка «верно или неверно» меняется только тогда, когда ответ переворачивается, так что почти всюду у неё нет наклона, по которому можно идти.
2. **a)** Обучение становится поиском параметров, при которых потери, усреднённые по набору данных, как можно меньше.
3. **b)** Отступ положителен, когда знак согласен с целью, и отрицателен, когда нет, и он меняется гладко.
4. **b)** Сети показывают сильное обобщение далеко за той точкой, где ожидалось переобучение, открывая новые направления исследований.

Задачи

1. На примере игры с монетой под чашками объясните, почему функция потерь должна быть дифференцируемой, чтобы градиентный спуск работал.

Подсказка: Сравните ответы «да или нет» с ответами «насколько далеко».

Ответ: С одним лишь «да» или «нет» неверная догадка ничего не говорит о том, куда двигаться, так что искателю приходится пробовать чашки наугад. С «насколько далеко» каждая догадка говорит, какое направление уменьшает расстояние.

Дифференцируемые потери похожи на ответ «насколько далеко»: их наклон говорит градиентному спуску, в какую сторону подстраивать параметры.

2. Опишите словами поведение кусочно-линейных потерь и объясните, почему они полезнее оценки «верно или неверно».

Подсказка: Что происходит, когда отступ достаточно велик, и когда он сжимается или становится неверным?

Ответ: Кусочно-линейные потери равны нулю, как только отступ достаточно велик, то есть модель явно права, и ровно растут по мере того, как отступ сжимается или становится отрицательным.

Поскольку они гладко меняются с отступом, у них есть наклоны, которые говорят градиентному спуску, как улучшаться, в отличие от оценки «верно или неверно».

Запоминание — не обучение

Представьте студента, который получает прошлогодний экзаменационный билет, заучивает все ответы и набирает полный балл на пробном прогоне по тому же билету. Потом приходит настоящий экзамен с новыми вопросами, и студенту нечего сказать. Пробный балл был идеальным и ничего не значил. Книга использует именно эту ловушку, чтобы показать, почему низкие средние потери на обучающих данных — не настоящая цель.

ОБУЧЕНИЕ: БЕЗ ОШИБОК

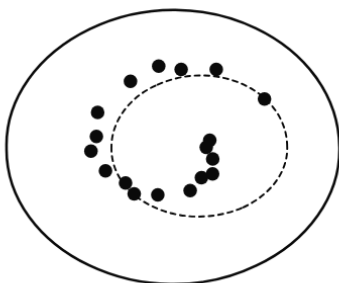
	ВХОД	ОТВЕТ	
● →	●	4	✓
	▲	7	
	■	2	
×	⬢ →	ПО УМОЛЧАНИЮ 0	

Таблица, которая только помнит

На самом деле средние потери на обучающих данных всегда тривиально легко довести до минимума. Постройте таблицу подстановки: если вход — один из обучающих примеров, вернуть его сохранённый ответ; иначе вернуть какое-то фиксированное значение по умолчанию, например ноль. Эта таблица никогда не ошибается на обучающих данных, так что она достигает наименьших возможных средних потерь, и всё же она совершенно бесполезна на практике. В этом разница между запоминанием и обучением.

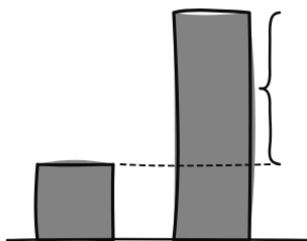
Что мы действительно хотим — низкие потери на будущих входах, которых ещё никто не видел; обучающие примеры лишь заменяют их. Книга называет это ожидаемым риском: средние потери по всем возможным парам вход-выход, с которыми модель может когда-либо встретиться, взвешенные по вероятности каждой, например по всем возможным электронным письмам. Модель с низким ожидаемым риском гарантированно работала бы хорошо, но его нельзя вычислить, потому что никто не может перечислить и разметить каждый возможный вход. Средние потери на наборе данных, прежний эмпирический риск, — его оценка, построенная из образцов, которые у нас оказались.

ВСЕ ВХОДЫ



ОБРАЗЦЫ

РАЗРЫВ



ОБУЧЕНИЕ ТЕСТ

Разрыв обобщения

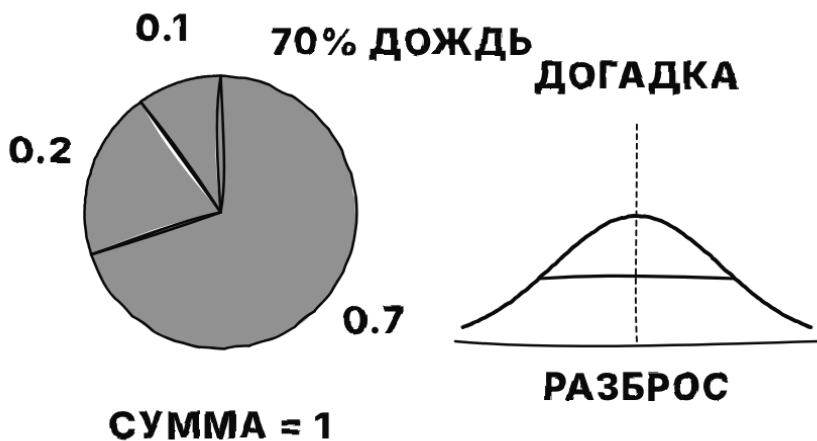
Разницу между ними называют разрывом обобщения. У запоминающей модели вроде таблицы подстановки разрыв огромен: она переобучается, подгоняясь под конкретные обучающие данные вместо закономерности за ними. Практическая проверка — держать отдельный тестовый набор, примеры, никогда не использованные при обучении, и сравнивать потери на обучающих данных с потерями на тестовом наборе. Большая разница — признак переобучения. Вот почему приложение для фотографий оценивают на снимках, которые ему никогда не показывали при обучении: только они раскрывают, выучило ли оно собак или просто запомнило определённые картинки.

Опыт

Попросите друга заучить ответы на десять вопросов викторины, а потом спросите его те же десять: он ответит идеально. Теперь задайте десять новых вопросов на те же темы. Второй балл, а не первый, говорит, сколько он действительно выучил, а падение между двумя баллами — собственный разрыв обобщения вашего друга.

Как выбирать функцию потерь? Книга предлагает сначала быстро просмотреть её приложение о вероятности, а потом объясняет, что потери несут наше предварительное знание о задаче и сильно влияют на качество. Иногда достаточно здравого смысла, чтобы её придумать, как с кусочно-линейными потерями. Но есть и более принципиальный путь: описать весь процесс обучения на языке вероятности, точка зрения, которую предпочитают многие другие книги.

Исходная идея в том, что примеры приходят из неизвестного источника пар вход-выход, который можно разбить на две части: насколько вероятен каждый вход и насколько вероятен каждый выход, когда вход известен. Методы, которые моделируют вероятность входов, называют генеративными, а те, что моделируют только выход при данном входе, — дискриминативными, и, если не считать языковых моделей, эта книга сосредоточена на втором виде. Предсказывать один единственный ответ имеет смысл, когда для каждого входа почти вся вероятность сидит на одном значении.



Ответы как распределения

Книга ослабляет это: вместо одного ответа выходы модели задают ручки целого распределения вероятностей над возможными ответами. Для выбора среди трёх классов модель даёт три вероятности, которые в сумме дают единицу, круговую диаграмму над классами. Для предсказания одного числа модель может дать два выхода: лучшую догадку и меру разброса, причём второй возводится в квадрат, чтобы он никогда не был отрицательным. Эта постановка включает всё сказанное до сих пор, и часто проще выбрать разумное распределение, чем изобретать потери вручную. Она также напрямую выражает неуверенность, через разброс. Приложение погоды, которое говорит «70 процентов вероятности дождя», а не просто «дождь», даёт вам ровно такой ответ.

Запомним

Таблица подстановки может достичь минимальных средних потерь на обучающих данных и быть бесполезной, что показывает: запоминание — не обучение. Важен ожидаемый риск, средние потери по всем возможным будущим данным; его нельзя вычислить, и эмпирический риск лишь оценивает его. Разница между ними — разрыв обобщения; большой разрыв означает переобучение, а отдельный тестовый набор его измеряет. В вероятностном взгляде модель выдаёт настройки распределения над ответами, например вероятности классов или догадку с её разбросом, что также выражает неуверенность.

Вопросы

1. Таблица подстановки достигает наименьших возможных средних потерь на обучающих данных. Почему она всё же бесполезна?
 - a. Потому что она слишком медленно работает
 - b. Потому что она лишь запоминает обучающие пары и даёт фиксированное значение по умолчанию для всего нового
 - c. Потому что её потери не дифференцируемы
 - d. Потому что для работы ей нужен тестовый набор

2. Почему нельзя просто обучать модель, напрямую минимизируя ожидаемый риск?
 - a. Потому что ожидаемый риск всегда равен нулю
 - b. Потому что для этого нужно перечислить и разметить каждый возможный вход, а это невозможно
 - c. Потому что он игнорирует функцию потерь
 - d. Потому что он работает только для электронных писем
3. У модели очень низкие потери на обучающих данных и гораздо более высокие на отдельном тестовом наборе. На что это указывает?
 - a. Модель идеально обобщает
 - b. В тестовом наборе наверняка ошибки
 - c. Модель переобучается, с большим разрывом обобщения
 - d. Функция потерь не дифференцируема
4. В вероятностном взгляде что выдаёт модель для предсказания одного числа вместо одного ответа?
 - a. Лучшую догадку вместе с мерой разброса
 - b. Три вероятности, которые в сумме дают единицу
 - c. Только значение потерь
 - d. Таблицу подстановки прошлых ответов

Ответы

1. **b)** Таблица возвращает сохранённые ответы для обучающих входов и значение по умолчанию для остальных, так что закономерность она не выучила: запоминание — не обучение.

- 2. б)** Ожидаемый риск усредняет по всем возможным парам вход-выход, которые никто не может перечислить; эмпирический риск лишь оценивает его.
- 3. с)** Большая разница между потерями на обучении и на тесте — практический признак большого разрыва обобщения, то есть переобучения.
- 4. а)** Для одного числа модель даёт лучшую догадку и разброс, который держат положительным возведением в квадрат, и это также выражает её неуверенность.

Задачи

1. Вы обучаете спам-фильтр и сообщаете его средние потери на письмах, на которых он обучался. Объясните, почему одно это число не может сказать, работает ли фильтр, и что следует сделать вместо этого.

Подсказка: Вспомните студента и прошлогодний экзаменационный билет.

Ответ: Потери на обучении можно сделать крошечными даже чистым запоминанием, так что они мало говорят о новых письмах. Держите отдельный тестовый набор писем, никогда не использованных при обучении, и измеряйте потери на нём; разница с потерями на обучении оценивает разрыв обобщения и раскрывает переобучение.

2. Объясните своими словами, почему вероятностный взгляд может быть полезнее модели, которая даёт один единственный ответ.

Подсказка: Вспомните приложение погоды и выбор потерь.

Ответ: Распределение над ответами, например вероятности классов или догадка с её разбросом, говорит, насколько модель уверена, как

вероятность дождя вместо плоского прогноза. Оно также включает прежнюю постановку как частный случай и упрощает устройство обучения, потому что выбрать разумное распределение часто проще, чем изобретать потери вручную.

Самая правдоподобная модель

Представьте, что трое друзей утверждают, что каждый подбросил монету двадцать раз, и один из них протягивает вам листок с восемнадцатью орлами. Один друг говорит, что использовал обычную монету, другой — монету со смещением к орлу, третий — монету со смещением к решке. Без всякой формулы вы уже знаете, чьей истории верить: монета со смещением к орлу делает этот листок куда более вероятным, чем остальные. Выбор объяснения, при котором наблюдаемые данные наиболее вероятны, — идея максимального правдоподобия, и именно так книга обучает вероятностные модели.



Какая монета объясняет 18 орлов?

Вспомните, что примеры в наборе данных считаются извлечёнными независимо из одного и того же неизвестного источника. Так что модель, которая даёт вероятности ответам, даёт вероятность и всему набору данных: шансы всех примеров, перемноженные между собой. Это число называют правдоподобием набора данных. Модель, выбранная случайно, распределяет свои вероятности почти слепо, поэтому наш конкретный набор данных она находит очень маловероятным. Разумный выбор — модель, при которой наш набор данных как можно более вероятен: решение максимального правдоподобия.

Это всё ещё задача оптимизации, но с одним отличием. Раньше функция потерь была проектным решением; здесь, как только

распределения вероятностей выбраны, цель напрямую следует из законов вероятности. И всё же два взгляда встречаются.

Логарифмирование превращает длинное произведение в сумму, а смена знака превращает максимизацию в минимизацию, так что максимальное правдоподобие становится минимизацией среднего из минус-логарифмов вероятностей, своего рода псевдопотерь. Книга добавляет, что каждую функцию потерь, используемую на практике, можно получить таким способом при подходящем выборе распределения, и призывает читателя держать в голове оба взгляда. Обучение чат-ассистента, который учится давать высокую вероятность настоящему следующему слову в своём обучающем тексте, — это максимальное правдоподобие в гигантском масштабе.

Затем книга идёт ещё на шаг дальше, к байесовскому обучению и байесовским нейронным сетям, давая лишь главную идею. Даже вероятностная модель после обучения остаётся одной единственной функцией, однако много разных настроек модели могли бы объяснять данные одинаково хорошо, и сохранение их всех могло бы сделать предсказания более честными. Рецепт снова в том, чтобы позволить вероятности вести нас. Сначала задать априорное распределение: насколько правдоподобной кажется каждая возможная функция до того, как увидены какие-либо данные. На практике это означает априорное распределение на весах, и распространённый выбор отдаёт предпочтение меньшим весам, потому что такие функции обычно устойчивее.



Априорное умножить на правдоподобие

Когда приходят данные, теорема Байеса соединяет априорное распределение с правдоподобием и даёт апостериорное: обновлённую правдоподобность каждой функции после того, как увидены данные; нормирующий член, называемый свидетельством, делает его настоящим распределением. Чтобы предсказывать, усредняют предсказания всех функций, каждую с весом её апостериорной правдоподобности, а на практике это приближают, выбирая горстку функций и усредняя их. Загвоздка в том, что апостериорное распределение почти никогда нельзя вычислить точно, поэтому нужны приближения, такими методами как марковские цепи Монте-Карло или вариационный вывод. Девятая глава покажет простой практический пример, приём под названием Монте-Карло-дропаут.

Опыт

Наполните банку шариками двух цветов в пропорции, которую вы не раскрываете, и попросите кого-нибудь угадать пропорцию до того, как он увидит хоть один шарик: эта догадка — его априорное распределение. Теперь дайте ему вытаскивать по одному шарiku и обновлять догадку после каждого. Заметьте, что каждая обновлённая догадка становится отправной точкой для следующего шарика: вчерашний вывод — сегодняшнее априорное распределение.

Раздел закрывают два замечания. Первое: если нам нужна лишь одна самая правдоподобная функция, можно не обращать внимания на свидетельство и максимизировать правдоподобие и априорное распределение вместе, что называют решением максимума апостериорной вероятности. Если априорное распределение относится ко всем функциям одинаково, это снова просто максимальное правдоподобие. Иначе априорное распределение штрафует функции, которые слишком далеко уходят от ожидаемого, и этот штраф, называемый регуляризатором, — полезное оружие против переобучения. Книга замечает, что разница примерно отражает два способа понимать вероятность: как частоту событий, считая параметры фиксированными, а данные случайными, или как меру неуверенности, считая данные фиксированными, а параметры неопределёнными.

Второе: байесовский взгляд делает естественным обучение на новых данных по мере их поступления. Апостериорное распределение, вычисленное на первой порции данных, просто

становится априорным для следующей порции, которая затем снова его обновляет. Это может облегчить известную проблему непрерывно обучаемых моделей, называемую катастрофическим забыванием, когда изучение новой информации стирает старую. Приложение, которое продолжает учиться на вашем поведении, сталкивается ровно с этим риском забыть выученное раньше, и такой способ обновлять убеждения — один из путей его смягчить.



Сегодняшнее апостериорное — завтрашнее априорное

Запомним

Правдоподобие — вероятность, которую модель даёт всему набору данных, а максимальное правдоподобие выбирает модель, при которой наблюдаемые данные наиболее вероятны. В логарифмической форме это становится минимизацией суммы минус-логарифмов вероятностей, и каждую практическую функцию потерь можно вывести таким способом. Байесовское обучение сохраняет целое распределение над функциями: априорное до данных, обновлённое теоремой Байеса в апостериорное, с предсказаниями, усреднёнными по правдоподобным функциям, хотя обычно лишь приближённо. Самая правдоподобная единственная функция, решение максимума апостериорной вероятности, добавляет априорное распределение как регуляризатор против переобучения, а апостериорное может служить априорным для новых данных в борьбе с катастрофическим забыванием.

Вопросы

1. Почему случайно выбранная модель даёт нашему набору данных очень низкое правдоподобие?
 - а. Потому что она распределяет свои вероятности почти слепо, так что наши конкретные примеры получают мало вероятности

- b. Потому что правдоподобия до обучения всегда равны нулю
 - c. Потому что случайные модели не умеют перемножать вероятности
 - d. Потому что набор данных собран не независимо
2. Как максимальное правдоподобие связано с функциями потерь, которые мы видели раньше?
- a. Оно заменяет градиентный спуск подсчётом
 - b. Логарифмирование и смена знака превращают его в минимизацию суммы минус-логарифмов вероятностей, псевдопотерь
 - c. Оно работает только с кусочно-линейными потерями
 - d. Оно никак не связано с потерями
3. В решении максимума апостериорной вероятности что происходит, если априорное распределение считает все функции одинаково правдоподобными?
- a. Оно становится простым решением максимального правдоподобия
 - b. Оно всегда выбирает наименьшие веса
 - c. Его нельзя вычислить
 - d. Оно полностью игнорирует данные
4. Как байесовский взгляд помогает модели продолжать учиться на новых данных, не забывая старые?
- a. Он удаляет старые данные после каждой порции
 - b. Апостериорное распределение по прежним данным становится априорным для новых

- с. Он обучает совершенно новую модель для каждой порции
- д. Он обнуляет все веса перед каждым обновлением

Ответы

- 1. а) Правдоподобие перемножает шансы, которые модель даёт всем примерам, а случайная модель даёт нашим конкретным примерам мало вероятности.
- 2. б) Логарифм превращает произведение в сумму, а смена знака превращает максимизацию в минимизацию, так что это становится обычной минимизацией потерь.
- 3. а) При равномерном априорном распределении априорный член постоянен, так что важно только правдоподобие, и решение — максимальное правдоподобие.
- 4. б) Вчерашний вывод становится сегодняшним априорным распределением, так что обновление строится на выученном, облегчая катастрофическое забывание.

Задачи

- 1. Объясните своими словами роли априорного распределения, правдоподобия и апостериорного распределения в байесовском обучении, используя банку с шариками.

Подсказка: Какое убеждение идёт до вытаскивания, какое приходит из вытаскиваний, а какое после?

Ответ: Априорное распределение — догадка о пропорции цветов до того, как вытащен хоть один шарик. Правдоподобие говорит, насколько вероятны вытащенные шарик при каждой возможной пропорции. Апостериорное распределение соединяет оба в

обновлённое убеждение после вытаскиваний, и оно может служить априорным для следующих вытаскиваний.

2. Почему книга называет априорный член в решении максимума апостериорной вероятности регуляризатором и почему это помогает против переобучения?

Подсказка: Подумайте, что априорное распределение делает с функциями, которые уходят от ожидаемого.

Ответ: Априорное распределение добавляет штраф для функций, которые далеко уходят от ожидаемого, например для функций с очень большими весами. Это толкает решение к более простым, более устойчивым функциям вместо тех, что подгоняются под каждую причуду обучающих данных, и это уменьшает переобучение.

3. Почему полное байесовское обучение трудно использовать точно на практике и что делают вместо этого?

Подсказка: Подумайте о вычислении апостериорного распределения.

Ответ: Апостериорное распределение над всеми функциями почти никогда нельзя вычислить точно, кроме особых выборов априорного распределения и правдоподобия. Поэтому используют приближения, такие как марковские цепи Монте-Карло или вариационный вывод, а предсказания усредняют по горстке выбранных функций.